情報工学実験Ⅰ

「オブジェクト指向プログラミング」

担当教員名：赤嶺 有平

提　出　日：２０１０年6月１７日
学 籍 番 号：０９５７０７B
氏　　名：大城　佳明

LEVEL 1

なぜOOPを利用するのか，なぜ必要とされたのか歴史的な背景を交えて考察せよ．

～２０年前～

①オブジェクト指向という概念がなかった。
②当時のコンピュータは性能が非常に低かった
③C言語などの低級言語で全てのコードが書かれていました。
④低レベルの言語でもコードを書くことができていた。
⑤大規模なプログラムは動かすこともできず、需要もなかった。

～現代～

①コンピュータは昔のコンピュータとCPUのクロック周波数やメモリなどに相当の差があり、大きな容量を利用している
②人間にとって効率のいいレベルで理解し易いプログラムを書く必要が出てきたからである
③構造化されたプログラムもわかり易いが構造化プログラムの場合、グローバル変数をC言語では対応させる必要がある

～必要性～

①オブジェクト指向プログラミングではグローバル変数の利用を抑制することができる。
②コード の再利用の幅を広げることができるので、一度ライブラリを構築してしまえばこのライブラリを使ってそれ以降のコーディングの効率を上げることができる。

なぜ、オブジェクト指向がこんなことをできるかというと、オブジェクト指向言語においては、アプリケーションごとに一つだけ存在している変数ではなく、オブジェクトと呼ばれる処理対象となっているものに対して値を割り当てることができるからである。このようなことから、構造化されたプログラムよりオブジェクト指向を利用したプログラムの方が効率がいいので、現在必要とされている。

LEVEL 2

本節(2 節）の各演習(2.1.1 – 2.6.3)におけるソースコードを考察せよ．特に変更点と改善点について述べよ．

演習 2.1.1

下記のコードは，glut を用いた Window の表示とユーザエベントハンドリングの例である．コンパイル，実行せよ

<glut_basic.c>

```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 #include <OpenGL/gl.h>
05 #include <GLUT/glut.h>
06
07 int window_x = 2;
08 int window_y = 2;
09
10 //描画イベントハンドラ
11 void display()
12 {
13        glClear(GL_COLOR_BUFFER_BIT); //背景を塗りつぶす．色は，glClearColor で指定しておく
14
15        glFinish(); //描画命令を実際に実行する．
16 }
17
18 //マウスイベントハンドラ
19 void mouse(int button, int state, int x, int y)
20 {
21        //マウスの座標系は，左上が(0,0), 右下が(window_x, window_y)である．
22        //OpenGL の座標系は，左上(-1,1),  右下（1,-1)なので，変換する必要がある．
23        float fx = (float)x / window_x * 2 - 1;
24        float fy = (float)-y / window_y * 2 + 1;
25
26        if(state == GLUT_UP) {
27
28                printf("mouse up");
29        }else{
30
31                printf("mouse down");
32        }
33
34        printf("pos(%f, %f)\n", fx, fy);
35
36        glutPostRedisplay();
37 }
38
39 //キーボードイベントハンドラ
40 void keyboard(unsigned char key, int x, int y)
41 {
42        if(key == 'q')
43                exit(0);
```

```
44
45        printf("keyboard(%c)\n", key);
46 }
47
48 //ウインドウサイズ変更ハンドラ
49 void resize(int sx, int sy)
50{
51        window_x = sx;
52        window_y = sy;
53
54        glViewport(0, 0, sx, sy);
55        printf("resize(%d,%d)\n", sx,sy);
56 }
57
58 //OpenGL 関係の初期化処理
60 void initializeOpenGL()
61 {
62        glClearColor(0, 0, 0, 0);
63 }
64
67 int main(int argc, char *argv[])
68 {
69        glutInit(&argc, argv); //glut の初期化
70        glutCreateWindow(argv[0]); //window の作成
71
72        //イベントハンドラの登録
73        glutDisplayFunc(display);  //描画コールバック関数の登録
74        glutMouseFunc(mouse);      //マウスクリックコールバック関数の登録
75        glutKeyboardFunc(keyboard);//キー入力コールバック関数の登録
76        glutReshapeFunc(resize);   //ウインドウサイズ変更コールバック関数の登録
77
78        //OpenGL 関係の初期化処理
79        initializeOpenGL();
80
81        //イベントループ(ユーザの操作を待つループ）に入る
82        glutMainLoop();
83
84        //ここが実行される事はない
85        printf("exit\n");
86        return 0;
87 }
```

<実行結果>

```
01 resize(300,300)
02 mouse downpos(-0.306667, 0.166667)
03 mouse uppos(-0.306667, 0.166667)
04 keyboard(a)
05 keyboard(e)
06 mouse downpos(-0.146667, 0.133333)
07 mouse uppos(-0.146667, 0.133333)
```

<考察>

1.04,05 はイベントハンドラを行うために必要なプログラムである

2.11～16 はディスプレイの色などを設定している

3.19～37 はマウスをクリックしたときの動作と座標を求めて、表示する

4.40～46 はキーボードで打った文字を表示する

5.49～56 ウィンドウの表示サイズを変更する

6.60～63 はコメントにも書かれているが、OpenGL 関係の初期化をおこなう

7.64～87 は main 関数である

A.72～76 はイベントハンドラの登録を行っている

B.ユーザが操作をすると、読み込まれる

C.イベントループがおこなわれるため、85,86 は読み込まれることはない

8.実行結果よりわかること。

A.実行すると、実行結果 01 と画面が表示された

B.ソースの 73,76 が実行されたことがわかる

C.実行結果 02,03,06,08 はマウスをクリックした時に表示された

D.ソースの 74 が実行されたことがわかる

F.04,05 はキーボード操作した時に表示された

演習２・１・２

glut_basic.c に以下を追加せよ

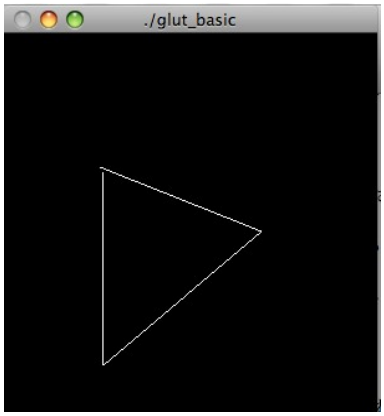
<glut_basic.c>追加

```
  …

007 #define MAX_POINTS 100
008
009 float position_x[MAX_POINTS];
010 float position_y[MAX_POINTS];
011
012 int num_points;
013
014 int window_x = 2;
015 int window_y = 2;

  …

018 void display()
019 {
020   glClear(GL_COLOR_BUFFER_BIT); //背景を塗りつぶす．色は，glClearColor で指定しておく
021
022   glBegin(GL_LINE_STRIP); //連続した線分の描画の開始を OpenGL に通知する
023
024   glColor3f(1,1,1); //以降に追加する頂点の色を指定(red,green,blue)
025
026   int i;
027   for(i=0; i<num_points; ++i) {
028     glVertex2f(position_x[i], position_y[i]); //頂点を追加
029   }
030
031   glEnd(); //線分描画の終了を OpenGL へ通知
032
033   glFinish(); //描画命令を実際に実行する．
034 }

  …

044  if(state == GLUT_UP) {
045    position_x[num_points] = fx;
046    position_y[num_points] = fy;
047
048     num_points ++;
049
050     printf("mouse up");
051  }else{

  …

097   //OpenGL 関係の初期化処理
098   initializeOpenGL();
099
```

```
100   //線分の数を初期化
101   num_points = 0;
102
103   //イベントループ(ユーザの操作を待つループ）に入る
104   glutMainLoop();

  …
```

<実行結果>追加

```
01 resize(300,300)
02 mouse downpos(-0.486667, 0.293333)
03 mouse uppos(-0.486667, 0.293333)
04 mouse downpos(0.380000, -0.046667)
05 mouse uppos(0.380000, -0.046667)
06 mouse downpos(-0.466667, -0.753333)
07 mouse uppos(-0.466667, -0.753333)
08 mouse downpos(-0.473333, 0.266667)
09 mouse uppos(-0.473333, 0.266667)
```



※「…」は省略を意味する

<考察>

１.追加したソースの解析

A.007～012 まで追加した

ⅰ.007 で「MAX_POINTS」を１００と宣言している

ⅱ.009,010 は座標をいれる配列である

ⅲ.012 は座標の数が入る

B.022～031 まで追加した

ⅰ.022,024 はコメントに書かれている

ⅱ.026～029 は座標の頂点を求め、一つ一つを線分で繋げることをやっている

ⅲ.031 は線分描画の終了を OpenGL へ通知をしている

C.045～048 まで追加した

ⅰ.045,046 は配列の num_points 番目にクリックされた座標を入れている

ⅱ.048 は num_points をインクリメントしている。

D.100,101 を追加した

ⅰ.最初の処理で、線分の数は１本目なので、「0」を入れて初期化を行う

２.実行結果よりわかること

A.３つクリックした

B.線分が３本できている

３.配列を使う事で、座標を数多く保存することが出来る

４.実行結果では、線分を１本ずつ追加しているように見えるが、実際は１回１回、１から描いている

演習２，２
前節のコードをモジュール化せよ

<module_gui.c>

```
  …

05 #include <GLUT/glut.h>
06 #include "module_line_struct.h"

  …

17 //描画イベントハンドラ
18 void display()
19 {
20   glClear(GL_COLOR_BUFFER_BIT); //背景を塗りつぶす．色は，glClearColor で指定しておく
21
22   drawLine();
23
24   glFinish(); //描画命令を実際に実行する．
25 }
26
27 //マウスイベントハンドラ
28 void mouse(int button, int state, int x, int y)
29 {
30   //マウスの座標系は，左上が(0,0), 右下が(window_x, window_y)である．
31   //OpenGL の座標系は，左上(-1,1),　右下（1,-1)なので，変換する必要がある．
32   float fx = (float)x / window_x * 2 - 1;
33   float fy = (float)-y / window_y * 2 + 1;
34
35   if(state == GLUT_UP) {
36
37     addPoint(fx,fy);
38
39     printf("mouse up");
40   }else{
41
42     printf("mouse down");
43   }
45


  …

84   //OpenGL 関係の初期化処理
85   initializeOpenGL();
86
87   //線分の数を初期化
88   initializeLine();
89
90   //イベントループ(ユーザの操作を待つループ）に入る
91   glutMainLoop();
```

```
…
```

<module_line.c>

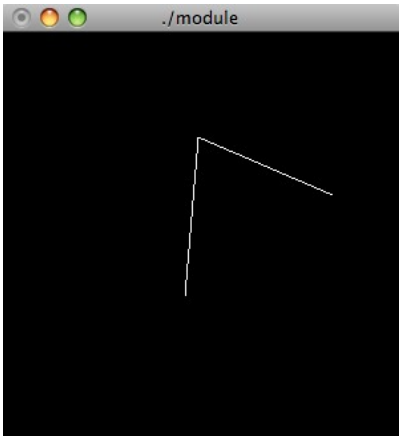
```
01 //module_line.c
02
03 #include <OpenGL/gl.h>
04
05 #define MAX_POINTS 100
06
07 float position_x[MAX_POINTS];
08 float position_y[MAX_POINTS];
09
10 int num_points;
11
12 void initializeLine()
13 {
14   //線分の数を初期化
15   num_points = 0;
16 }
17
18 void drawLine()
19 {
20   glBegin(GL_LINE_STRIP); //連続した線分を描画する
21
22   int i;
23   for(i=0; i<num_points; ++i) {
24     glVertex2f(position_x[i], position_y[i]);
25   }
26
27   glEnd();
28 }
29
30 void addPoint(float x, float y)
31 {
32   position_x[num_points] = x;
33   position_y[num_points] = y;
34
35   num_points ++;
36 }
```

<module_line.h>

```
01 //module_line.h
02
03 void initializeLine();
04 void drawLine();
05 void addPoint(float x, float y);
```

<実行結果>

```
01 esize(300,300)
02 mouse downpos(-0.080000, -0.306667)
03 mouse uppos(-0.080000, -0.306667)
04 mouse downpos(-0.013333, 0.480000)
05 mouse uppos(-0.013333, 0.480000)
06 mouse downpos(0.660000, 0.193333)
07 mouse uppos(0.660000, 0.193333)
```



※「…」は省略を意味する

<考察>

1.追加したソースの解析をする

A.06 を追加した

ⅰ.”module_line_struct.h”を読み込んでいる

ⅱ.同じフォルダにある場合はダブルフォーてション(“)で囲む

B.22 を追加した。

ⅰ.module_line_stuct.h にある関数を利用している

ⅱ.線を描く関数である

C.37 を追加した

ⅰ.module_line_struct.h にある関数を利用している

ⅱ.引数はクリックしたときの座標である

D.88 を追加した

ⅰ.module_line_struct.h にある関数を利用している

ⅱ.線分の初期化を行う関数である

2.module_line.c を解析する

A.03～10 は必要な引数の宣言である

B.12～16 は線分の数を初期化するための関数である

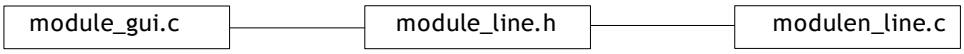
C.18～28 は線分を描くための関数である

D.30～36 は座標を配列にいれるための関数である

3.module_line.h を解析する

A.すべて関数の宣言である

B.module_line.c と module_gui.c を繋げるためのファイルである



4.実行結果は変わっていないことがわかる

5.機能毎にソースコードが分割されモジュール化された

6.モジュール化することによって、関数を別で定義する事によりソースを簡単にすることができる

演習 2.3
module_gui_struct.c から module_line_struct.h をインクルードするように変更せよ
演習 2.3.1
頂点と線分の構造体を module_line_struct.h に定義せよ

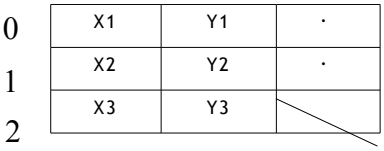
<module_line_struct.h>

```
01 //module_line_struct.h
02 #define MAX_POINTS 100
03
04 struct Point {
05   float x;
06   float y;
07 };
08
09 struct Line {
10   struct Point points[MAX_POINTS];
11   int numPoints;
12 };
13
14 void initializeLine();

  …
```

<考察>

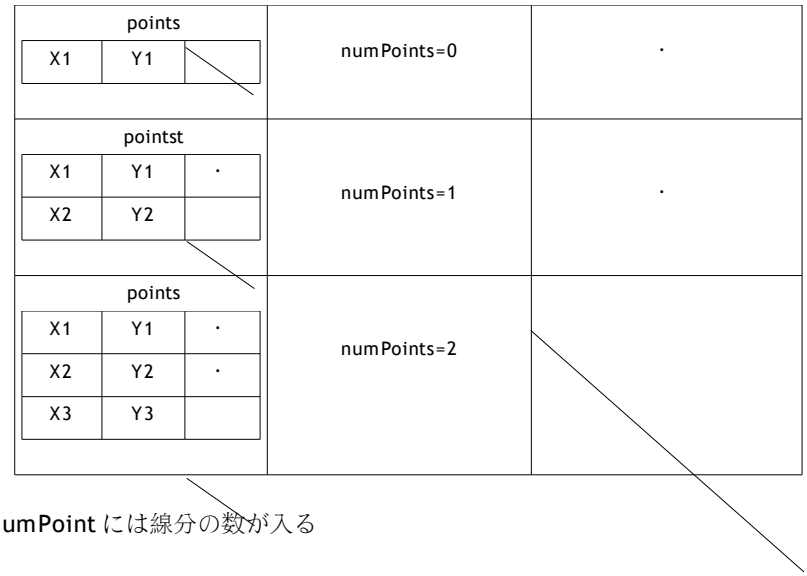
1.04～07 は Point という構造体の宣言をしている

struct Point



2.09 では Line という構造体の宣言をしている

struct Line



3.numPoint には線分の数が入る

4.したがって、線分が 1 本の時、 2 本の時、 … 、n 本の時までのそれぞれの線が保存されている

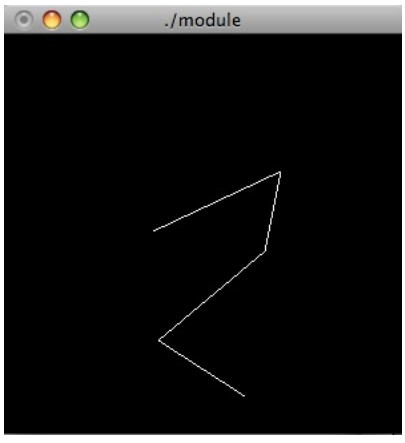
演習 2.3.2
module_line_struct.c を構造体を用いたコードに変更せよ

<module_line_struct.c>

```
01 //module_line_struct.c
02
03 #include <OpenGL/gl.h>
04 #include "module_line_struct.h"
05
06 struct Line a_line;
07
08 #define MAX_POINTS 100
09
10 void initializeLine()
11 {
12   //線分の数を初期化
13   a_line.numPoints = 0;
14 }
15
16 void drawLine()
17 {
18   glBegin(GL_LINE_STRIP); //連続した線分を描画する
19
20   int i;
21   for(i=0; i < a_line.numPoints; ++i) {
22     glVertex2f(a_line.points[i].x, a_line.points[i].y);
23   }
24
25   glEnd();
26 }
27
28 void addPoint(float x, float y)
29 {
30   a_line.points[a_line.numPoints].x = x;
31   a_line.points[a_line.numPoints].y = y;
32
33   a_line.numPoints ++;
34 }
```

<実行結果>

```
01 resize(300,300)
02 mouse downpos(-0.253333, 0.013333)
03 mouse uppos(-0.253333, 0.013333)
04 mouse downpos(0.386667, 0.313333)
05 mouse uppos(0.386667, 0.313333)
06 mouse downpos(0.306667, -0.086667)
07 mouse uppos(0.306667, -0.086667)
08 mouse downpos(-0.226667, -0.533333)
09 mouse uppos(-0.226667, -0.533333)
10 mouse downpos(0.206667, -0.813333)
11 mouse uppos(0.206667, -0.813333)
```



<考察>

１.追加、変更したソースの解析をする

A.module_line_struct.c の追加、変更したところについて

B.06 では struct Line を a_line にして宣言している

C.13 では numPoints を a_lime の構造体から取得している。それに「０」を入れている

D.同様に 21,22,30,31,33 も構造体から取得している

E.配列からの引数を、すべて構造体からの引数にしている

２.module_gui_struct.c は変更する事がなかった

３.実行結果は変わってないことがわかる

４.構造体の形にする事によって、オブジェクト指向プログラミングの準備となる。

演習 2.4.1
initializeLine, drawLine, addPoint の各関数の引数に Line 構造体へのポインタを追加せよ

<module_line_object.c>

```
01 void initializeLine(struct Line* l)
02 {
03   //線分の数を初期化
04   l->numPoints = 0;
05 }
06
07 void drawLine(struct Line* l)
08 {
09   glBegin(GL_LINE_STRIP); //連続した線分を描画する
10
11   glColor3f(1,1,1); //以降に追加する頂点の色を指定(red,green,blue)
12
13   int i;
14   for(i=0; i < l->numPoints; ++i) {
15     struct Point p = l->points[i];
16     glVertex2f(p.x, p.y);
17   }
18
19   glEnd();
20 }
21
22 void addPoint(struct Line* l,float x, float y)
23 {
24   l->points[l->numPoints].x = x;
25   l->points[l->numPoints].y = y;
26
27   l->numPoints ++;
28 }
```

<module_gui_object.c>

```
  …

06 #include "module_line_object.h"
07
08 struct Line a_line;
09
10 #define MAX_POINTS 100

  …

16 void display()
17 {
18   glClear(GL_COLOR_BUFFER_BIT); //背景を塗りつぶす．色は，glClearColor で指定しておく
19
20   drawLine(&a_line);
21
```

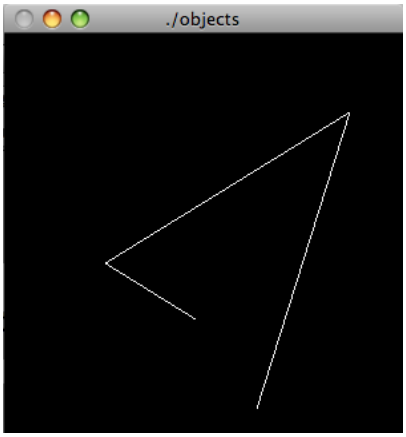
```
22   glFinish(); //描画命令を実際に実行する．
23 }

   …

33   if(state == GLUT_UP) {
34
35     addPoint(&a_line,fx,fy);
36
37     printf("mouse up");

83   //線分の数を初期化
84   initializeLine(&a_line);

   …
```

<module_line_object.h>

```
   …

14 void initializeLine(struct Line* l);
15 void drawLine(struct Line* l);
16 void addPoint(struct Line* l,float x, float y);
```

<実行結果>

```
01 resize(300,300)
02 mouse downpos(-0.040000, -0.426667)
03 mouse uppos(-0.040000, -0.426667)
04 mouse downpos(-0.493333, -0.146667)
05 mouse uppos(-0.493333, -0.146667)
06 mouse downpos(0.733333, 0.606667)
07 mouse uppos(0.733333, 0.606667)
08 mouse downpos(0.266667, -0.873333)
09 mouse uppos(0.266667, -0.873333)
```



<考察>

1.ソースの変更、追加の考察をする

A.module_line_object.c について考察する

ⅰ.オブジェクト指向にするためにグローバル変数を消す

ⅱ.「struct Line a_line」を削除した

ⅲ.04 の「a_line.numPoints」を「l->numPoints」のように値からポインタへ変更する

ⅳ.同様に 14,24,25,27 も値からポインタへ変更した

ⅴ.01,07,22 では関数にポインタの引数の受け渡しを行うために「struct Line* l」を加えた

ⅵ.15 では l->ponts[i]を p にすることで 16 を見やすくしている

B.module_gui_object.c について考察する

ⅰ.08 の「struct Line a_line」を追加した

ⅱ.20 では drawLine 関数の引数「&a_line」が追加された

ⅲ同様に 35,84 も引数が追加されている

C.module_line_object.h について考察する

ⅰ.すべての関数に「struct Line* l」が追加された

2.module_line_object.c のグローバル変数を消す事により、ソースがオブジェクト指向となる

3.グローバル変数を消す代わりに、外部からの引数を受け取る必要となった。

4.ポインタの引数を受け取る事によって、数多くの構造体を扱うことが出来るようになった

演習 2.5
複数の線分を扱えるように変更せよ
<module_gui_objects.c>

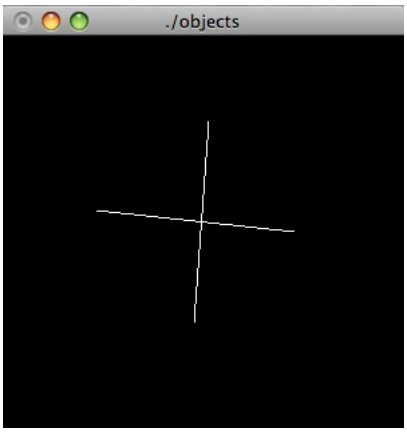
```
001 #include <stdio.h>
002 #include <stdlib.h>
003
004 #include <GLUT/glut.h>
005 #include <OpenGL/gl.h>
006
007 #include "module_line_object.h"
008
009 #define MAX_LINES 1000
010
011 struct Line lines[MAX_LINES];
012 int num_lines = 1;
013
014 int window_x = 2;
015 int window_y = 2;
016
017
018 //描画イベントハンドラ
019 void display()
020 {
021     glClear(GL_COLOR_BUFFER_BIT); //背景を塗りつぶす．色は，glClearColor で指定しておく
022
023     int i;
024     for(i=0; i<num_lines; ++i) {
025         drawLine(lines + i);
026     }
026
027     glFinish(); //描画命令を実際に実行する．
028 }
029
030 //マウスイベントハンドラ
031 void mouse(int button, int state, int x, int y)
032 {
033     float fx = (float)x / window_x * 2 - 1;
034     float fy = (float)-y / window_y * 2 + 1;
035
036     if(state == GLUT_UP) {
037         printf("mouse up\n");
038     }else{
039         addPoint(lines+(num_lines-1), fx, fy);
040         printf("mouse down\n");
041     }
042
043     glutPostRedisplay();
044 }
045
046 //キーボードイベントハンドラ
047 void keyboard(unsigned char key, int x, int y)
```

```
048 {
049     if(key == 'q')
050             exit(0);
051
052     if(key == ' ') {
053             num_lines++;
054             initializeLine(lines + (num_lines-1));
055     }
056
057     printf("keyboard(%c)\n", key);
058 }
059 //ウインドウサイズ変更ハンドラ
060 void resize(int sx, int sy)
061 {
062     window_x = sx;
063     window_y = sy;
064
065     glViewport(0, 0, sx, sy);
066     printf("resize(%d,%d)\n", sx,sy);
067 }
068
069 //OpenGL 関係の初期化処理
070 void initializeOpenGL()
071 {
072     glClearColor(0, 0, 0, 0);
073 }
074
075
076
077 int main(int argc, char *argv[])
078 {
079     glutInit(&argc, argv); //glut の初期化
080     glutCreateWindow(argv[0]); //window の作成
081
082     //イベントハンドラの登録
083     glutDisplayFunc(display);  //描画コールバック関数の登録
084     glutMouseFunc(mouse);      //マウスクリックコールバック関数の登録
085     glutKeyboardFunc(keyboard);//キー入力コールバック関数の登録
086     glutReshapeFunc(resize);   //ウインドウサイズ変更コールバック関数の登録
087
088     //OpenGL 関係の初期化処理
089     initializeOpenGL();
090
091     //lines の最初の要素の初期化処理
092     initializeLine(lines);
093
094     //イベントループ(ユーザの操作を待つループ）に入る
095     glutMainLoop();
096
097     //ここが実行される事はない
098     printf("exit\n");
```

```
099       return 0;
100 }
```

<実行結果>

```
01 resize(300,300)
02 mouse down
03 mouse up
04 mouse down
05 mouse up
06 keyboard( )
07 mouse down
08 mouse up
09 mouse down
10 mouse up
```



<考察>

1 .module_gui_objects.c の解析をする

A.011 より「a_line」が「lines[MAX_LINES]」という配列になった

B.配列になったことにより、線が最大 1000 本引けるようになった

C.012 は lines の配列のカウンタである

D.num_lines は線の本数になる

E.023～026 は配列を最初から順に drawLine 関数にいれている

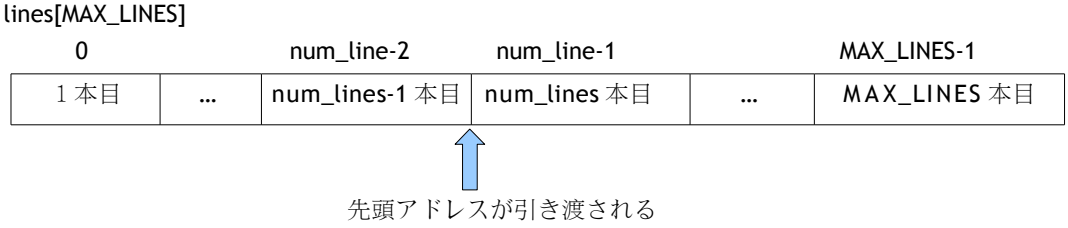
F.039 の addPoint は lines の num_lines-1 番目の配列の値を格納する

G.049 は「q」のキーを押すと、終了する

H.052～055 は「　」(スペース)キーを押すと、新しい線分が出来る

ⅰ.053 では num_lines をインクリメントして、次の配列に変わる

ⅱ.054 では lines の num_lines-1 番目の配列に格納する



H.092 は最初だけの処理なので lines の先頭の値を「０」にする必要がある

２.実行結果より２本出来るようになっている

３.配列を使うことによって、線がたくさん引けるようになった

４.module_line_object.c と module_line_object.h の中を変更しなくても、数多くの線が引けるようになった

５.これはオブジェク指向プログラムのおかげである

６.オブジェクト指向により、プログラム内も簡単に出来ている

演習 2.6.1
C++のクラス定義を実験せよ

<int.cpp>

```
01 #include <stdio.h>
02
03 //クラスの宣言
04 class Integer
05 {
06 public:
07   Integer();  //コンストラクタ、一般に初期化処理を行う
08   void setNumber(int n); //メソッド
09   int getNumber();//メソッド
10
11 private:
12   //インスタンス毎に,インスタンス変数の記憶領域が用意される
13   //すなわち，インスタンス変数は，各インスタンスで独立している
14   int number; //インスタンス変数（メンバ変数）
15 };
16
17 //コンストラクタの定義
18 Integer::Integer()
19 {
20   printf("constructor\n");
21   number = 1;
22 }
23
24 void Integer::setNumber(int n)
25 {
26   printf("setNumber\n");
27   number = n;
28 }
29
30 int Integer::getNumber()
31 {
32   printf("getNumber\n");
33   return number;
34 }
35
36 int main(int argc, char** argv)
37 {
38   printf("start main\n");
39
40   Integer a_integer;
41   Integer a_integer2;
42   Integer a_integer3;
43
44   a_integer.setNumber(5);
45   a_integer2.setNumber(10);
46
47   printf("a_integer = %d\n", a_integer.getNumber() );
```

```
48   printf("a_integer2 = %d\n", a_integer2.getNumber() );
49   printf("a_integer3 = %d\n", a_integer3.getNumber() );
50
51   return 0;
52 }
```

<実行結果>

```
01 start main
02 constructor
03 constructor
04 constructor
05 setNumber
06 setNumber
07 getNumber
08 a_integer = 5
09 getNumber
10 a_integer2 = 10
11 getNumber
12 a_integer3 = 1
```

<考察>

1.04～15 までがクラスの宣言である

A.04 より class Integer をいうクラスを宣言している

ｉ.06 の public とは「パブリック変数・関数はどのルーチンからもアクセス可能であることを示す。」

B.07 の Integer();はコンストラクタの宣言である

ｉ.コンストラクタとは「クラスをインスタンス化したときに、自動的に呼び出される特別なメンバ関数です。」

C.08,09 はメソッドの宣言である

D.11～14 より privete には int number が設定された

2.18～22 はコンストラクタの定義である

A.20 より constructor を表示する

B.21 より number には「1」が入る

3.24～28 は setNumber(int n)のメソッドの宣言である

A.26 より setNumber を表示する

B.27 より number の値を渡された数字「n」に変更する

4.30～33 は setNumber()のメソッドの宣言である

A.32 より getNumber と表示する

B.33 よりそのままの値を返す（つまり「１」である）

５.36～52 の main 関数の説明をする

A.38 では start main を表示する

B.40～42 はそれぞれコンストラクタを使っている

C.44 は a_integer を setNumber を使って number の値を「5」に変更している

D.45 も同様に「10」に変更している

E.47～49 はそれぞれの number の値を出力している

５.実行結果 02～04 より costructor が３回呼び出されていることがわかる

６.実行結果 05,06 ではソース 44,45 の setNumber(int n)による処理である

７.実行結果 08,10 より値が指定通りに変更されていることがわかる

８.実行結果 12 よりコンストラクタの呼び出しにより、初期状態（number=1）であることがわかる

演習 2.6.2
前節のコードを C++ で記述せよ

<gui_with_lines.cpp>

```
  …

005 #include <OpenGL/gl.h>
006
007 #include "line.h"

  …

011 Line lines[MAX_LINES];
012 int num_lines = 1; //line の数

  …

023   int i;
024   for(i=0; i<num_lines; ++i) {
025     lines[i].draw();
026   }

  …

039   }else{
040     lines[num_lines-1].addPoint(fx, fy);
041    printf("mouse down\n");

  …

053   if(key == ' ') {
054     num_lines++;
055   }

  …

088
089   //lines の最初の要素の初期化処理
090  //初期化はコンストラクタで実行するため不要

  …
```

<line.cpp>

```
01 #include <OpenGL/gl.h>
02
03 #include "line.h"
04
05 Line::Line()
06 {
07   numPoints = 0;
```

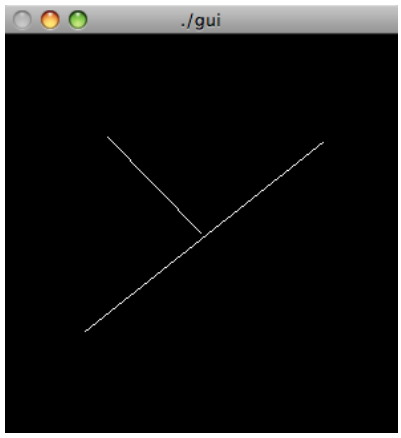
```
08 }
09
10 void Line::draw()
11 {
12   glBegin(GL_LINE_STRIP); //連続した線分の描画の開始を OpenGL に通知する
13
14   int i;
15   for(i=0; i<numPoints; ++i) {
16     struct Point p = points[i];
17
18     glColor3f(1, 1, 1);   //頂点の色を指定
19     glVertex2f(p.x, p.y); //頂点を追加する
20   }
21
22   glEnd(); //線分の描画が終了した事を OpenGL に通知する
23 }
24
25 void Line::addPoint(float x, float y)
26 {
27   points[numPoints].x = x;
28   points[numPoints].y = y;
29
30   numPoints ++;
31 }
```

<line.h>

```
01 #define MAX_POINTS 10000
02
03 struct Point {
04   float x;
05   float y;
06 };
07
08 class Line {
09  public:
10   Line();
11
12   void draw();
13   void addPoint(float x, float y);
14  private:
15   struct Point points[MAX_POINTS];
16   int numPoints;
17 };
```

<実行結果>

```
01 resize(300,300)
02 mouse down
03 mouse up
04 mouse down
05 mouse up
06 keyboard( )
07 mouse down
08 mouse up
09 mouse down
10 mouse up
11 mouse down
12 mouse up
```



<考察>

1.ソースの解析をする。

A.gui_with_lines.cpp の解析をする

ⅰ.011 は Line のクラスを lines の配列に置いている

ⅱ.025 では lines の i 番目の Line の中にある draw 関数を読み込んでいる

ⅲ.040 も同様に line の num_lines-1 番目の Line の中にある addPoint 関数を読み込んでいる

lines[MAX_LINES]

1本目	2本目	…	(num_lines)本目	…
0	1	…	num_lines - 1	…

(line.h) Line

```
void addPoint(float x, float y);
```

(line.c) void Line::addPoint(float x, float y)

```
points[numPoints].x = x;
points[numPoints].y = y;
 numPoints ++;
```

ⅳ.054 では num_line をインクリメントして、違う線に変えるている

B.line.cpp の解析をする

ⅰ.05～08 はコンストラクタである

ⅱ.07 では初期化を行っている

ⅲ.10～23 は draw とういメソッドの宣言である

ⅳ.draw は Line のクラスなので「point[MAX_POINTS]」「numPints」が存在している

ⅴ.25～31 は addPoint というメソッドの宣言である

ⅵ.構造体とほとんど一緒の扱いでできる

C.line.h の解析をする

ⅰ.03～06 は構造体の宣言である

ⅱ.08～17 は Line のクラス宣言である

ⅲ.10 はコンストラクタで、12,13 はメソッドである

ⅳ.15,16 は各メソッドでの用意されている引数のことである

2.実行結果より同じように出来ていることが分かる

3.C++ではクラスを C の構造体と同じように使うことが出来てる

4.クラスは構造体よりも簡単に書くことが出来る

5.C++ではオブジェクト指向が簡単に出来る

演習 2.6.3
線分の管理をユーザインターフェースに関する処理を完全に分離せよ

<gui.cpp>

```
  …

05 #include <OpenGL/gl.h>
06
07 #include "tool.h"
08
09 Tool a_tool;

  …

17   glClear(GL_COLOR_BUFFER_BIT); //背景を塗りつぶす．色は，glClearColor で指定しておく
18
19   a_tool.draw(); //線分の描画命令を発行する
20
21  glFinish(); //描画命令を実際に実行する．

  …

34   }else{
33     a_tool.addPoint(fx, fy);
35     printf("mouse down\n");
36   }

  …

46   if(key == ' ') {
47     a_tool.makeLine();
48   }

  …

82   //OpenGL 関係の初期化処理
83   initializeOpenGL();
84
85   //最低一つは line が必要なので作成しておく
86   a_tool.makeLine();
87
88   //イベントループ(ユーザの操作を待つループ）に入る
89   glutMainLoop();

  …
```

<tool.cpp>

```
01 #include <OpenGL/gl.h>
02
03 #include "tool.h"
```

```
04
05 Tool::Tool()
06 {
07   numLines = 0;
08 }
09
10 void Tool::draw()
11 {
12   int i;
13   for(i=0; i<numLines; ++i) {
14     lines[i].draw();
15   }
16 }
17
18 void Tool::makeLine()
19 {
20   numLines++;
21 }
22
23 void Tool::addPoint(float x, float y)
24 {
25   lines[numLines-1].addPoint(x,y);
26 }
```

<tool.h>

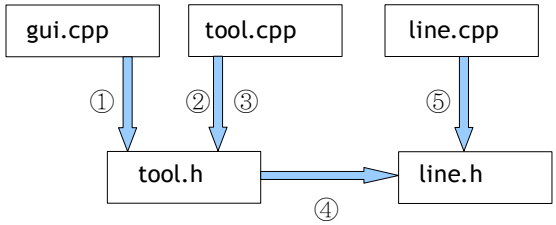
```
01 #include "line.h"
02
03 #define MAX_LINES 1000
04
05 class Tool {
06  public:
07   Tool();
08
09   void draw();
10   void makeLine();
11   void addPoint(float x, float y);
12
13  private:
14   Line lines[MAX_LINES];
15   int numLines;
16 };
```

<実行結果>

演習 2.6.2 と同じ実行結果になった

<考察>

１.プログラムの仕組みを図に表した



①gui.cpp から tool.h を使って tool クラスの呼び出しを行っている

②tool.h から tool.cpp の中のメソッドを読み出し、処理をする

③tool.cpp から tool.h を使って Line クラスの呼び出しを行っている

④too.h から line.h の構造体と関数を利用してする

⑤line.h から line.cpp に行き関数を使っている

２.tool.cpp と too.h を作り、２段階のオブジェクトを作っている

３.オブジェクト指向とモジュール化を徹底している

５.そのため ine.cpp 及び line.h はそのまま再利用可能である

４オブジェクト化したため、.gui.cpp のソースが簡単になった

参考文献

オブジェクト指向
http://ja.wikipedia.org/wiki/オブジェクト指向

C++の基礎知識
http://www.geocities.co.jp/HeartLand/6978/cpp.htm

C++編（言語解説） 第7章 コンストラクタとデストラクタ
http://www.geocities.jp/ky_webid/cpp/language/007.html