```
// ***************************************************
// ***** TD Question 1
// ***************************************************

struct Complex
{
    float x,y;
};


// ***************************************************
// ***** TD Question 2
// ***************************************************

Complex make_complex(float r, float i)
{
    Complex c;
    c.x = r;
    c.y = i;
    return c;
}

Complex make_complex_expo(float r, float theta)
{
    Complex c;
    c.x = r*cos(theta);
    c.y = r*sin(theta);
    return c;
}


// ***************************************************
// ***** TD Question 3
// ***************************************************

Complex operator+(Complex a, Complex b)
{
    Complex c = { a.x+b.x, a.y+b.y };
    return c;
}

Complex operator-(Complex a, Complex b)
{
    Complex c = { a.x-b.x, a.y-b.y };
    return c;
}

Complex translate(Complex p, float dx, float dy)
{
    return p + make_complex(dx,dy);
}


// ***************************************************
// ***** TD Question 4
// ***************************************************
//A( 1,-1) = 1 -1.i  = 1-i
//B( 0, 1) = 0 +1.i  = i
//C(-1,-1) = -1 -1.i = -1-i
//
//lambda=2
//A' = lambda * A = 2*(1-i) = 2-2i   = (2,-2)
//B' = lambda * B = 2*i              = (0, 2)
//C' = lambda * C = 2*(-1-i) = -2-2i = (-2,-2)
// ==> faire le dessin
//
//lambda=0.5
//A' = lambda * A = (0.5, -0.5)
//B' = lambda * B = (0, 0.5)
//C' = lambda * C = (-0.5, -0.5)
// ==> faire le dessin
```

```
72. //
73. // Multiplication d'un complexe par un rÃ©el = une homothÃ©tie de centre O = approche (
    lambda<1) ou Ã©loigne (lambda>1) le point
74.
75.
76.
77. // ************************************************
78. // ***** TD Question 5
79. // ************************************************
80.
81. Complex operator*(float a, Complex b)
82. {
83.     Complex c = { a*b.x, a*b.y };
84.     return c;
85. }
86. Complex operator/(Complex b, float d)
87. {
88.     Complex c = { b.x/d, b.y/d };
89.     return c;
90. }
91.
92. Complex scale(Complex p, float cx, float cy, float sc)
93. {
94.     Complex tr = make_complex( cx, cy);
95.     return (sc*(p-tr))+tr;
96. }
97.
98.
99.
100.      // ************************************************
101.      // ***** TD Question 6 et 7
102.      // ************************************************
103.
104.      // A( 1,-1) = 1 -1.i  = 1-i
105.      // B( 0, 1) = 0 +1.i  = i
106.      // C(-1,-1) = -1 -1.i = -1-i
107.      //
108.      // r = e^(i.theta) avec theta = PI/2 donc une rotation de 90Â°
109.      // r = cos(PI/2) + i.sin(PI/2) = 0+i = i
110.      //A' = r * A = i*(1-i) = 1+i    = (1,1)
111.      //B' = r * B = i*i = -1         = (-1, 0)
112.      //C' = r * C = i*(-1-i) = 1-i   = (1,-1)
113.      // ==> faire le dessin
114.      // ==> rotation de 90Â°
115.      //
116.      // ==> ca marche avec n'importe quel theta
117.      // par exemple avec theta=0  r= cos(0)+i.sin(0) = 1 => identite
118.      // par exemple avec theta=PI r= cos(PI)+i.sin(PI) = -1
119.      //
120.      // MultipliÃ© un complexe par un complexe imaginaire pure (r=e^(i.theta) ==> rota
    tion de thetaÂ°
121.
122.
123.
124.      // ************************************************
125.      // ***** TD Question 8
126.      // ************************************************
127.
128.      float to_degree(float rad)
129.      {
130.          return 180.f * rad/M_PI;
131.      }
132.
133.      float to_rad(float deg)
134.      {
135.          return M_PI*deg/180.f;
136.      }
137.
138.      Complex operator*(Complex a, Complex b)
139.      {
140.          Complex c = make_complex( a.x*b.x - a.y*b.y, a.x*b.y + a.y*b.x );
```

```
141.          return c;
142.      }
143.
144.      Complex rotate(Complex p, float cx, float cy, float theta_deg)
145.      {
146.          Complex rot = make_complex_expo( 1, to_rad(theta_deg));
147.          Complex tr = make_complex( cx, cy);
148.          return ((p-tr)*rot)+tr;
149.      }
150.
151.
152.      float norm(Complex c)
153.      {
154.          return sqrt( c.x*c.x + c.y*c.y);
155.      }
156.
157.
158.      // ************************************************
159.      // ***** TD Question 9
160.      // ************************************************
161.      // ************************** SOLAR SYSTEM ************************************
   **************
162.      struct SolarSystem
163.      {
164.          Complex sun;
165.          Complex mars;
166.          Complex earth;
167.          Complex moon;
168.
169.      };
170.
171.      void init(SolarSystem& ss)
172.      {
173.          ss.sun = make_complex(DIMW/2, DIMW/2);
174.          ss.mars = ss.sun + make_complex(30, 0);
175.          ss.earth = ss.sun + make_complex(80, 0);
176.          ss.moon = ss.earth + make_complex(15, 0);
177.      }
178.
179.      void draw(SolarSystem ss)
180.      {
181.          color(255,255,0);
182.          circleFill( ss.sun.x, ss.sun.y, 10);
183.
184.          color(255,25,0);
185.          circleFill( ss.mars.x, ss.mars.y, 4);
186.
187.          color(0,25,255);
188.          circleFill( ss.earth.x, ss.earth.y, 5);
189.
190.          color(125,25,155);
191.          circleFill( ss.moon.x, ss.moon.y, 2);
192.      }
193.
194.      void update(SolarSystem& ss)
195.      {
196.          ss.mars = rotate( ss.mars, ss.sun.x, ss.sun.y, .1f);        // Mars tourne au
   tour du soleil d'un angle 0.1 degrÃ© Ã  chaque frame
197.
198.          Complex moon_local = ss.moon - ss.earth;                   // Position de la
   lune par rapport Ã  la terre
199.          ss.earth = rotate( ss.earth, ss.sun.x, ss.sun.y, .03f);     // Terre tourne a
   utour du soleil d'un angle 0.03 degrÃ© Ã  chaque frame
200.          ss.moon = ss.earth + moon_local;                            // On remet la lu
   ne au mÃªme endroit par rapport Ã  la nouvelle position de la terre
201.
202.          ss.moon = rotate( ss.moon, ss.earth.x, ss.earth.y, .03f);   // Lune tourne au
   tour de la terre d'un angle 0.03 degrÃ© Ã  chaque frame
203.      }
204.
205.
```

```
206.      // ************************************************
207.      // ***** TD Question 10 Ã  17
208.      // ************************************************
209.      // ============================  POLYGON ============================================
    =====
210.      const int POLY_MAX_POINT = 100;
211.      struct Polygon
212.      {
213.          Complex p[POLY_MAX_POINT];
214.          int n;
215.      };
216.
217.      Complex CenterOfGravity(Polygon& p)
218.      {
219.          int i;
220.          Complex cog = make_complex(0,0);
221.          for(i=0;i<p.n;++i)
222.          {
223.              cog = cog + p.p[i];
224.          }
225.          return cog/p.n;
226.      }
227.
228.      void scale(Polygon& p, float sc)
229.      {
230.          Complex cog = CenterOfGravity(p);
231.          int i;
232.          for(i=0;i<p.n;++i)
233.          {
234.              p.p[i] = scale( p.p[i], cog.x, cog.y, sc);
235.          }
236.      }
237.
238.      void rotate(Polygon& p, float angle)
239.      {
240.          Complex cog = CenterOfGravity(p);
241.          int i;
242.          for(i=0;i<p.n;++i)
243.          {
244.              p.p[i] = rotate( p.p[i], cog.x, cog.y, angle);
245.          }
246.      }
247.
248.      void init(Polygon& po)
249.      {
250.          po.n = 0;
251.      }
252.
253.      void addPoint(Polygon& po, float x, float y)
254.      {
255.          if (po.n>=POLY_MAX_POINT) return;
256.          po.p[ po.n ] = make_complex( x, y);
257.          ++po.n;
258.      }
259.
260.      void print_poly(const Polygon& p)
261.      {
262.          cout<<"n="<<p.n<<endl;
263.          for(int i=0;i<p.n;++i)
264.              cout<<p.p[i].x<<", "<<p.p[i].y<<endl;
265.      }
266.
267.      void draw_polygon(Polygon& po)
268.      {
269.          int i,in;
270.          color(255,0,0);
271.          for(i=0;i<po.n;++i)
272.          {
273.              in = (i+1)%po.n;
274.              line( po.p[i].x, po.p[i].y, po.p[in].x, po.p[in].y );
275.          }
```

```
276.
277.         if (isMousePressed(SDL_BUTTON_RIGHT))
278.         {
279.             int x,y;
280.             mousePos(x, y);
281.
282.             if ((po.n==0) || (norm(po.p[po.n-1]-make_complex(x,y))>5.f))
283.                 addPoint(po, x,y);
284.         }
285.         if (isKeyPressed(SDLK_UP))
286.             scale(po, 1.2);
287.         if (isKeyPressed(SDLK_DOWN))
288.             scale(po, 0.8);
289.
290.         if (isKeyPressed(SDLK_LEFT))
291.             scale(po, -1.f);
292.
293.         if (isKeyPressed(SDLK_RIGHT))
294.             rotate(po, M_PI/6);
295.     }
```