```cpp
#include <algorithm>
#include <iostream>
#include <list>
#include <vector>

#include <cppqt.h>

using namespace std;

struct Kante
{
  int ymax; // maximale y-Koordinate
  double x; // AKTUELLE x-Koordinate
  double einsdurchm; // Da die Kanten in y-Richtung ausgewertet werden,
                     // wird die inverse Steigung benötigt.

  friend bool operator<(const Kante& k1, const Kante& k2)
  {
    if (k1.x != k2.x)
      return k1.x < k2.x;
    else
      return k1.einsdurchm < k2.einsdurchm;
  }
};

typedef list<Kante> KantenTabelle;

void drawFilledPolygon(Drawing& pic, const vector<IPoint2D>& ecken,
                       int colour = 0)
{
//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
  int ymin = pic.getHeight() + 1;
  int ymax = -1;
  unsigned int n = ecken.size();
```

```
35   cout<<ymin<<"  "<<ymax<<"  "<<endl;
36
37   // bestimme kleinste und größte Bildzeile
38   for (unsigned int i = 0; i < n; i++)
39      {
40        if (ecken[i].y < ymin)
41          ymin = ecken[i].y;
42        if (ecken[i].y > ymax)
43          ymax = ecken[i].y;
44      }
45
46   cout<<ymin<<"  "<<ymax<<"  "<<endl;
47
48   KantenTabelle aet(0);
49
50
51   vector<KantenTabelle> et(ymax + 1);             // et(0, …, ymin) bleiben leer
52   // Iteratoren
53   KantenTabelle::iterator kitaet;        // Kanteniterator für Active-Edge-Table
54   KantenTabelle::iterator kitety;      // Kanteniterator für Edge-Table, Zeile y
55   Kante k;
56   int kymin;
57   double xanf, xend;
58
59   /*
60    * Erzeuge die Kanten
61    */
62   for (unsigned int i = 0; i < n - 1; ++i)
63     {
64       if (ecken[i].y != ecken[i + 1].y)
65         {
66           if (ecken[i].y < ecken[i + 1].y)
67             {
68               k.ymax = ecken[i + 1].y;
```

```
69              k.x = ecken[i].x;
70              kymin = ecken[i].y;
71            }
72          else
73            {
74              k.ymax = ecken[i].y;
75              k.x = ecken[i + 1].x;
76              kymin = ecken[i + 1].y;
77            }
78          k.einsdurchm = static_cast<double>(ecken[i + 1].x - ecken[i].x) /
79            (ecken[i + 1].y - ecken[i].y);
80          et[kymin].push_back(k); //Füge die Kante in die zu ihrer y-Koordinate gehörenden Liste ein.
81        }
82    }
83  /*
84   * Das Polygon wird geschlossen, indem der letzte und der erste Punkt
85   * verbunden werden.
86   */
87  if (ecken[n - 1].y != ecken[0].y)
88    {
89      if (ecken[n - 1].y < ecken[0].y)
90        {
91          k.ymax = ecken[0].y;
92          k.x = ecken[n - 1].x;
93          kymin = ecken[n - 1].y;
94        }
95      else
96        {
97          k.ymax = ecken[n - 1].y;
98          k.x = ecken[0].x;
99          kymin = ecken[0].y;
100       }
101     k.einsdurchm = static_cast<double>(ecken[0].x - ecken[n - 1].x) /
102       (ecken[0].y - ecken[n - 1].y);
```

```
103       et[kymin].push_back(k);
104     }
105
106   /*
107    * Sortiere die Kanten in et[y] horizontal (siehe struct Kante).
108    * So lassen sich die Kanten einfach sortiert in die AET einfügen.
109    */
110   for (int y = ymin; y <= ymax; ++y)
111     et[y].sort();
112
113
114   /*
115    * Hauptschleife über die y-Zeilen des Polygons
116    */
117   for (int y = ymin; y <= ymax; ++y)
118     {
119       // füge alle in Zeile y startenden Kanten sortiert ein
120       kitety = et[y].begin();
121       kitaet = aet.begin();
122       while (kitety != et[y].end() && kitaet != aet.end())
123         {
124           if (*kitaet < *kitety)
125             ++kitaet;
126           else
127             {
128               aet.insert(kitaet, *kitety);
129               ++kitety;
130             }
131         }
132       /*
133        * Am Ende sind u.U. noch Kanten in et[y] übrig. Aufgrund der
134        * Sortierung können sie hier einfach angehängt werden.
135        */
136       while (kitety != et[y].end())
```

```
137         aet.push_back(*kitety++);
138
139       /*
140        * Entferne alle bei y endenden Kanten aus der AET.
141        */
142       for (kitaet = aet.begin(); kitaet != aet.end(); )
143         if (kitaet->ymax == y)
144           kitaet = aet.erase(kitaet);
145         else
146           ++kitaet;
147
148       /*
149        * Malen!
150        */
151       for (kitaet = aet.begin(); kitaet != aet.end(); )
152         {
153           xanf = (kitaet++)->x;
154           xend = (kitaet++)->x;
155           cout<<" Zeile y=  " <<y<<"    xanf und xend:     "<<xanf<<"    "<<xend<<endl;
156           for (int x = static_cast<int>(round(xanf));
157                x <=  static_cast<int>(round(xend)); ++x)
158             pic.drawPoint(x, y, colour, false);
159           cout<<"----------------------------------------"<<endl;
160         }
161       /*
162        * Aktualisiere die x-Werte aller Kanten in AET.
163        */
164       for (kitaet = aet.begin(); kitaet != aet.end(); ++kitaet)
165         kitaet->x += kitaet->einsdurchm;
166     }
167 //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
168 }
169
170 int maindraw( )
```

```
171 {
172   Drawing pic1(300, 300);
173   pic1.setSleepTime(3);
174 
175   pic1.show();
176   pic1.setZoom(2);
177 
178   int n;
179   IPoint2D p;
180   vector<IPoint2D> ecken;
181   // zum STL-Typ vector vergleiche z.B.:
182   //http://en.cppreference.com/w/cpp/container/vector
183   //http://www.cplusplus.com/reference/vector/vector/
184   //http://www.sgi.com/tech/stl/Vector.html
185 
186   cout << "Anzahl der Ecken: ";
187   cin >> n;
188 
189   for (int i = 1; i <= n; ++i)
190     {
191       cout << "Ecke " << i << ": ";
192       cin >> p;
193       ecken.push_back(p);
194     }
195 
196   cout << "Eingegebenes Polygon: ";
197   for (unsigned int i = 0; i < ecken.size(); ++i)
198     cout << ecken[i] << '-';
199   cout << ecken[0] << endl;
200 
201   drawFilledPolygon(pic1, ecken, 0);
202 
203   pic1.savePNG("polygon.png");
204 
```

```
205   cout << endl;
206   IOThread::waitForWindow(5);
207
208   return 0;
209 }
210
211 /*
212  4
213  (0,100)
214  (40,250)
215  (200, 200)
216  (50,0)
217   */
218
219
```