```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();// die Anzahl der Ecken des Polygons.

// durchlaufen wir die Scheitelpunkte, um die minimalen und maximalen y-Koordinaten des Polygons zu
finden.
for (unsigned int i = 0; i < n; i++)
{
  if (ecken[i].y < ymin)
    ymin = ecken[i].y;
  if (ecken[i].y > ymax)
    ymax = ecken[i].y;
}
/*
 *Initialisierung der Active Edge Table (aet), einer Liste von aktiven Kanten
 * für die aktuelle Scanline (initialisiert mit 0).
 */
KantenTabelle aet(0);
/*
 * Initialisierung der Kantentabelle (et), einen Vektor mit Listen von Kanten
 * für jede y-Koordinate.
 * Sie speichert alle Kanten des Polygons sortiert nach den y-Koordinaten.
 */
vector<KantenTabelle> et(ymax + 1);             // et(0, …, ymin) bleiben leer
// Iteratoren
KantenTabelle::iterator kitaet;        // Kanteniterator für Active-Edge-Table
KantenTabelle::iterator kitety;      // Kanteniterator für Edge-Table, Zeile y
Kante k; //Temporäre Variable zum Speichern einer Kante.
int kymin;  //TV zum Speichern der minimalen y-Koordinate einer Kante.
double xanf, xend;//TV zum Speichern der Anfangs- und End-X-Koordinaten einer Spanne.
```

```
/*
 * Erzeuge die Kanten aus der Punkteliste. Wichtig
 * 1. sortieren wir die beiden Punkte einer Kante nach ihrer y-Koordinate.
 * 2. Die aktuelle x-Koordinate ist dann zunächst die x-Koordinate des unteren Punktes.
 * 3. Wir fügen die Kante entsprechend ihrer y-Koordinate in die Liste ein.
 */
for (unsigned int i = 0; i < n - 1; ++i)
  {
    if (ecken[i].y != ecken[i + 1].y)//Ist die y-Koordinate der aktuellen und der nächsten Ecke
    unterschiedlich? was pasiert wenn die gleich sind?
      {//Wenn ja, werden die maximale y-Koordinate und die Start-x-Koordinate der Kante bestimmt.
        if (ecken[i].y < ecken[i + 1].y)
          {
            k.ymax = ecken[i + 1].y;
            k.x = ecken[i].x;  //(2)
            kymin = ecken[i].y;
          }
        else
          {
            k.ymax = ecken[i].y;
            k.x = ecken[i + 1].x;
            kymin = ecken[i + 1].y;
          }//Dann wird die inverse Steigung (k.einsdurchm) berechnet und die Kante zur
          Kantentabelle hinzugefügt.
        k.einsdurchm = static_cast<double>(ecken[i + 1].x - ecken[i].x) /
          (ecken[i + 1].y - ecken[i].y);
        et[kymin].push_back(k);  //|3|6|8|new|
      }
  }
```

```
/*
 * Das Polygon wird geschlossen, indem der letzte und der erste Punkt
 * verbunden werden.
 */
if (ecken[n - 1].y != ecken[0].y)
  {// wir machen das gleiche wie vorher. aber für die erste und letzte Ecke
    if (ecken[n - 1].y < ecken[0].y)
      {
        k.ymax = ecken[0].y;
        k.x = ecken[n - 1].x;
        kymin = ecken[n - 1].y;
      }
    else
      {
        k.ymax = ecken[n - 1].y;
        k.x = ecken[0].x;
        kymin = ecken[0].y;
      }
    k.einsdurchm = static_cast<double>(ecken[0].x - ecken[n - 1].x) /
      (ecken[0].y - ecken[n - 1].y);
    et[kymin].push_back(k);
  }

/*
 * Wir sortieren die Kanten in et[y] horizontal (siehe struct Kante).
 * So lassen sich die Kanten einfach sortiert in die AET einfügen.
 */
for (int y = ymin; y <= ymax; ++y)
  et[y].sort(); //Die Sortierung basiert auf dem Vergleich, der in der Struktur kante
```

```
// NICE! Hauptschleife über die y-Zeilen des Polygons
for (int y = ymin; y <= ymax; ++y)
  {
    // fügen wir alle Kanten, die in der Zeile y beginnen, sortiert ein.
    kitety = et[y].begin();
    kitaet = aet.begin();

    /*Wir starten eine while-Schleife, die so lange läuft, bis entweder
     * alle Kanten an der aktuellen y-Koordinate in der Kantentabelle oder
     * alle Kanten in der aktiven Kantentabelle verarbeitet sind.*/
    while (kitety != et[y].end() && kitaet != aet.end())
      {//Wenn die aktuelle Kante in der AET "kleiner" ist als die aktuelle Kante in der ET
        if (*kitaet < *kitety)
          ++kitaet; // dann wird zur nächsten Kante in der Tabelle der aktiven Kanten gewechselt.
        else//Andernfalls
          {//fügen wir die Kante aus der ET (*kitety) an der aktuellen Position (kitaet) in die AET
          ein.
            aet.insert(kitaet, *kitety);
            ++kitety; //Dann gehen wir zur nächsten Kante in der Kantentabelle.
          }
      }
    /*
     * Alle verbleibenden Kanten in et[y] werden an die Tabelle
     * der aktiven Kanten (aet) angehängt, da sie bereits sortiert sind.
     */
    while (kitety != et[y].end())
      aet.push_back(*kitety++);
```

```
    /* Wir entfernen alle bei y endenden Kanten aus der AET.
     * Das Entfernen ist notwendig, um die Spans zwischen Paaren
     * von Kanten der AET malen zu können. Dieser Fall tritt immer dann
     * ein, wenn am Ende einer Kante beim aktuellen y eine neue Kante
     * beginnt.  Zum Besibiel y+2
     */
    for (kitaet = aet.begin(); kitaet != aet.end(); )
      if (kitaet->ymax == y)
        kitaet = aet.erase(kitaet);
      else
        ++kitaet;
    /*
     * Male Pixel der einzelnen Spans. Spans werden immer
     * zwischen Paaren von Kanten aus der AET gemalt.
     */
    for (kitaet = aet.begin(); kitaet != aet.end(); ){
        xanf = (kitaet++)->x;// wir nehmen x,  und dann inkrementiert den Iterator
        xend = (kitaet++)->x;
        for (int x = static_cast<int>(round(xanf));
             x <=  static_cast<int>(round(xend)); ++x)
          pic.drawPoint(x, y, colour, false);
      }
    /*
     * Aktualisiere die x-Werte aller Kanten in AET.
     */
    for (kitaet = aet.begin(); kitaet != aet.end(); ++kitaet)
      kitaet->x += kitaet->einsdurchm;// x= x+1/m
  }
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
}


int maindraw( )
{
  Drawing pic1(300, 300);
  pic1.setSleepTime(3);

  pic1.show( );
  pic1.setZoom(2);

  int n;
  IPoint2D p;
  vector<IPoint2D> ecken;
  // zum STL-Typ vector vergleiche z.B.:
  //http://en.cppreference.com/w/cpp/container/vector
  //http://www.cplusplus.com/reference/vector/vector/
  //http://www.sgi.com/tech/stl/Vector.html

  cout << "Anzahl der Ecken: ";
  cin >> n;

  for (int i = 1; i <= n; ++i)
    {
      cout << "Ecke " << i << ": ";
      cin >> p;
      ecken.push_back(p);
    }

  cout << "Eingegebenes Polygon: ";
```

```
  for (unsigned int i = 0; i < ecken.size(); ++i)
    cout << ecken[i] << '-';
  cout << ecken[0] << endl;

  drawFilledPolygon(pic1, ecken, 0);

  pic1.savePNG("polygon.png");

  cout << endl;
  IOThread::waitForWindow(5);

  return 0;
}

/*
 4
 (0,100)
 (40,250)
 (200, 200)
 (50,0)
  */
```