-VisualC++ 2008 による-

LatticePointCounter の作成

浦田 敏夫

目次

1　Pick の定理と Windows アプリケーション

Pick の定理は，格子点を頂点とする多角形 P の辺周上の格子点の数 p と内部の格子点の数 n はこの多角形 P の面積 S とつぎの様な関係があると主張している：

$$S \ = \ n \ + \ \frac{p-2}{2} \ ,$$

ここでいう格子点とは整数座標 (x, y) を持つ平面上の点のことである．

明らかに，格子点を頂点とする多角形 P の辺周上の格子点と内部の格子点は，この多角形 P が格子も含めて紙上に書かれていれば，（一目瞭然）点を眼で追って探し出すことができ，この多角形 P の辺周上の格子点の数 p と内部の格子点の数 n を知ることは容易な事である．従って，算数教育の一話題ともなり得るのである．

Pick の定理を知れば，‘ 格子点を頂点とする任意の平面多角形の辺周上の頂点の情報からその辺周上および内部の格子点の位置を知る ’ 算術的手続きに興味が湧く．

格子点を頂点とする多角形 P の頂点の情報を与えて，この多角形 P の辺周上の格子点の数と内部の格子点の数を知るプログラムは簡単にかけるだろうか？

かって 1991 年頃に，この Pick の定理を話題にした時に，(格子点を頂点とする) 任意の平面多角形の辺周上の格子点数と内部の格子点数を数えるプログラムを BASIC で書き，また MSDOS 上では Borland C++ で書いた．

この手記では，平面上の格子点を結ぶ線分や格子点を頂点とする多角形（その辺周が平面を内部と外部に分割する（数学的には，Jordan 曲線））に注目している．

多角形内の格子点数を数えるための アプリケーション LatticePCounter の仕様

アプリケーション LatticePCounter は，マウスの左クリックで (格子点である) 頂点 $P_1, P_2, \cdots, P_{n-1}$ を拾い，右クリックで (格子点である) 頂点 P_n を拾い頂点 P_1 へ結んで閉じて，隣接する線分からなる多角図形 (の辺周) P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ とした後，多角図形 P の辺周上の格子点 (数) と内部の格子点 (数) を探し出すものであって，つぎの情報を獲得する：

- マウスで取り出した多角図形 P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ の辺周上の格子点をプロットし数える．
- 多角図形 P が多角形であるかどうか判定する．
- 多角形の場合に，内部の格子点をプロットし数える．

注意すべきは，（マウスを使って）順序に取得した頂点列を（この順序で）結んでできる多角図形 P が多角形になるとは限らない事である．とは言っても，有限個の順序付けられた頂点の入力情報から，“ 多角図形 P が平面を内部と外部に分割する多角形（数学的には，Jordan 曲線）となっている ”かどうかの判定は，頂点 $P_1, P_2, \cdots, P_n$ の重複を禁止し，線分 $P_1P_2, P_2P_3, \cdots, P_nP_1$ の互いの交差や線分の重なりを調べて判定できる．さらに，この多角図形 P が多角形となる場合には，（必要なら，頂点の順序を逆にすることで）辺周上の頂点は，正の向き（多角形の内部に対して反時計回りに向きづけられている）であるようにできる．

多角形内の格子点数を求めるためのよく知られている手続きの一つは，格子点を頂点とする多角形 P は一つの長方形 R の中に存在するのであるから，多角形 P を含む長方形 R の中の格子点のすべてについて，多角形 P の辺周上にあるのかまたは多角形 P の内部にあるかを走査的に調べることとされている．長方形 R の中の格子点 v が多角形 P に含まれるか否かは，格子点 v を端点とする半直線が多角形 P の辺周と交わる回数を，線分 $P_1P_2, P_2P_3, \cdots, P_nP_1$ との交差を線分と半直線の重なりに注意して数えて判定できるのだから，求めるプログラムを書くことは簡単であろう．残る興味は，実行に当たっての計算量や所要時間が問題になるだけか？

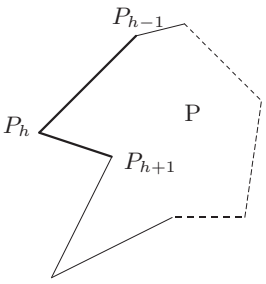
さて，我々は他の道があることに気づいた．「格子点を頂点とする多角形の内部の格子点は，この多角形の頂点が与えられればそれらから整数演算で容易に記述される.」という事実は，次に述べる定理からもわかる：

多角図形の Horn（角）

多角図形 P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ の頂点の座標を $P_i = (x_i, y_i)$ $(i = 1, 2, \cdots, n)$ とするときの，最左翼最高頂点における角を多角図形 P の Horn と呼ぶ．すなわち，

$$\text{条件} \quad \begin{cases} x_h \leq x_i & (i = 1, \cdots, n) \\ x_h = x_i \implies y_h > y_i & (i = 1, \cdots, n) \end{cases}$$

を満たす頂点 P_h における角を多角図形 P の Horn と呼ぶ．この場合，必要ならば頂点 $P_1, P_2, \cdots, P_n$ の番号付けを逆にして，三角形 $\triangle P_{h-1}P_hP_{h+1}$ の三頂点は正の向き（三角形の内部に対して反時計回りに向き，参照 補題）となるようにしておく．このとき，
頂点 P_h における角 $\angle P_{h-1}P_hP_{h+1}$ が多角図形 P の Horn を示す．



補題．多角図形の Horn での向き付け
多角図形 P の Horn $\angle SAB$ を考える; S, A, B は多角図形 P の頂点である．
$\overrightarrow{AB} = (x_1, y_1) \in \mathbf{Z}^2$ とすると， $x_1 \geq 0$ である． $\overrightarrow{SA} = (x_0, y_0) \in \mathbf{Z}^2$ とおくと， $x_0 < 0$.
$\overrightarrow{AB}$ を頂点 A の周りに角 $\dfrac{\pi}{2}$ 回転した $\overrightarrow{\boldsymbol{u}} = (-y_1, x_1)$ に対して， $\overrightarrow{\boldsymbol{u}}$ と $-\overrightarrow{SA}$ のなす角が $-\dfrac{\pi}{2}$ より大きく $\dfrac{\pi}{2}$ より小さいための必要充分条件は

$$-\overrightarrow{SA} \cdot \overrightarrow{\boldsymbol{u}} = x_0 \cdot y_1 - y_0 \cdot x_1 > 0$$

である．

この補題の状況下で， $-\overrightarrow{SA} \cdot \overrightarrow{\boldsymbol{u}} = x_0 \cdot y_1 - y_0 \cdot x_1 < 0$ の場合には，多角図形 P の頂点番号付けを逆にして多角図形 P の Horn での正の向きと決める．もし多角図形 P が多角形であれば，辺周上の頂点のこの向き付けは多角形の正の向きに合致する．

定理１．多角形 P の Horn $\angle SAB$ を考える; S, A, B は多角形 P の頂点である．
多角形 P の頂点 S, A, B の順序は多角形 P の正の向きに合っている．この時、以下のことが成り立つ．
(1)　辺 AB 上の格子点 $E\ (\neq A)$ を線分 AE 上には A, E 以外の格子点が存在しない様に取り，
$\overrightarrow{AE} = (x_1, y_1) \in \mathbf{Z}^2$ とする．このとき，$x_1 \geq 0$ で最大公約数 GCD $(x_1, y_1) = 1$ であり，方程式

$$x_1 \cdot y - y_1 \cdot x = 1 \quad (x, y \in \mathbf{Z})$$

の１つの整数解 $(cx, cy) \in \mathbf{Z}$ が存在する；$\overrightarrow{AD} = (cx, cy) \in \mathbf{Z}^2$ とする．
格子点 D は格子点 A, E, B を通る直線 $l_0 : t\overrightarrow{AE}\quad (\forall t \in \mathbf{R})$ に平行な直線 l_1

$$l_1 \quad : \quad t\overrightarrow{AE} + \overrightarrow{AD} \qquad (\forall t \in \mathbf{R})$$

上にある（格子点 A を原点 $\boldsymbol{O}$ として考えても，この定理の主張に問題は生じない！）．

(2)　辺 SA 上の格子点 $R\ (\neq A)$ を，線分 RA 上には A, R 以外の格子点が存在しない様に取り，

$$\overrightarrow{RA} = (x_0, y_0) \in \mathbf{Z}^2$$

とおくと，　$x_0 < 0$.
このとき　$x_1 \cdot cy - y_1 \cdot cx = 1$ であるから，連立方程式

$$\begin{cases} -x_0 = x_1 \cdot x + cx \cdot y \\ -y_0 = y_1 \cdot x + cy \cdot y \end{cases}$$

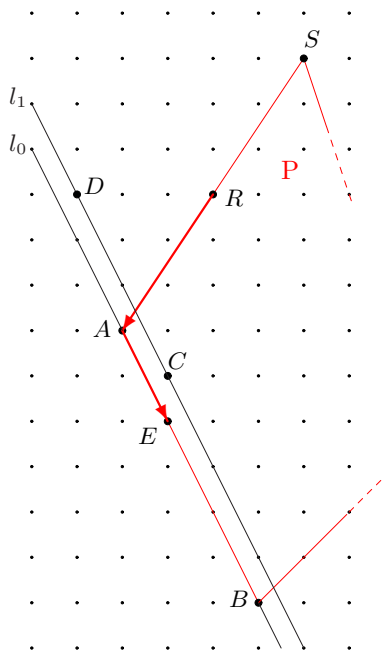
は整数解 $x = px,\ y = py$ を持ち，

$$\begin{cases} px = -x_0 \cdot cy + y_0 \cdot cx \\ py = -x_1 \cdot y_0 + y_1 \cdot x_0 \ > 0 \end{cases}$$

が成り立つ．

$$m = \begin{cases} px & (py = 1 \text{ の場合}) \\ \left[\dfrac{px}{py}\right] + 1 & (py > 1 \text{ の場合}) \end{cases}$$

と置く．
（ここで，Gauss 記号　$[a] = \max\{n \in \mathbf{Z} \ : \ n \leq a\}$.）



この時，$\overrightarrow{AC} = m\overrightarrow{AE} + \overrightarrow{AD}$ となる格子点 C に対して，$\triangle AEC$ は多角形 P に含まれる面積 $\dfrac{1}{2}$ の三角形で，その内部と辺上には（頂点以外）格子点がない (証明は付録，参考文献 [3]).

(*) 注意 1. 定理１．において，格子点 C は多角形 P の辺周上の格子点であることもある．
(*) 注意 2. 定理１．において，$x_1 = 0$ のとき $y_1 = -1$ である．この場合には, $cx = 1,\ cy = 0$ とすることができて，$px = y_0,\ py = -x_0 > 0$ が成り立つ．

この多角形の Horn を利用して，直接的に多角形内の格子点を発見する手続きを実行することを考える:

定理１．において，多角形 P : $P_1P_2, P_2P_3, \cdots, P_nP_1$ の隣接する辺 SA, AB を 隣接する線分 SA, AC, CE, EB で置き換えて多角形 Q が得られる．多角形 Q の面積は P の面積 - $\frac{1}{2}$ であるから，この手続きを

$$\text{P の面積} \div \frac{1}{2} = 2 \times \text{P の面積}$$

回を繰り返せば，多角形 P に含まれるすべての格子点を知ることができる．さらに，(格子点を頂点とする) 任意の平面多角形の辺周上の格子点数と内部の格子点数を知ることができる．

ここでは，このことに基づいて，多角図形 P (が多角形である場合に，その内部の格子点であるはず) の格子点をすべて求めつつ，同時に多角図形 P が多角形であるかどうかを判定できるプログラムを作成したい．

そこで，**格子点数を数えるための アプリケーション LatticePCounter の仕様** を多角形の判定を含む **仕様　格子多角形の取得** と **仕様　格子多角形の内点の取得** の二段階に分割した．

仕様　格子多角形の取得

- マウスの左クリックで多角図形の頂点 $P_1, P_2, \cdots, P_{n-1}$ を拾い，右クリックで頂点 P_n を拾い頂点 P_1 へ結んで辺周として閉じて，多角図形 P : $P_1P_2, P_2P_3, \cdots, P_nP_1$ とする．
- マウスで取り出した多角図形 (辺周) P : $P_1P_2, P_2P_3, \cdots, P_nP_1$ が多角形であるためのいくつかの基本的事項，例えば 頂点の重畳，辺の重畳，辺の交差等を調べて多角図形の頂点の情報を取得する．
- 辺周上の格子点をプロットし，辺周上の格子点数を確認する．

仕様　格子多角形の内点の取得

- 多角形 P : $P_1P_2, P_2P_3, \cdots, P_nP_1$ 内の格子点をの位置を取得し，数え上げる．
- 多角形 P 内の格子点をプロットする．

アプリケーション LatticePCounter は，多角図形が多角形であると仮定して多角形内の格子点を取得する手続きを通して，内部に格子点が在りえない三角形 (面積 $\frac{1}{2}$ である) への縮小から，この手続きの完了と多角図形が多角形であることを確かめる；　この手続きが完了せず（矛盾的状況で）終了することは，この多角図形が多角形でないことを示すものである．

こうして，格子点を頂点とする任意の平面多角形 P の頂点の情報からその辺周上および内部の格子点を数えるプログラム LatticePCounter を BASIC で書くことができ，また MSDOS 上では Borland C++ で書いてみることもできた．その後，Windows 上では [1] を参考にして，Visual C++ 6.0 で Windows 上に移植することができた．ところが，最近，上記のプログラム LatticePCounter を Visual C++ 2008 上で作成しなおそうとしたら，具体的な手続きがわからなかったので（素人の悲しさ），心覚えにこれを書いている．

2 Visual C++ 上での アプリケーション LatticePCounter の作成

VC++ で MFC（MicroSoft Foundation Class) を利用して Windows プログラムを作成しようする (私のような) 素人にとっては，AppWizard や ClassWizard の項目を選択することおよびイベントハンドラーに表示・操作方法のコードを書くことだけでも，具体的にどうしたらよいのかを学ぶことが易しくない．

考えてみると，C 言語で書かれたプログラムでの出発点 int main() が見えないし，C 言語や C++ のオブジェクト指向言語に関する知識の使い方もわからない．VC++ の参考書を探して，やっとクラスは，MFC クラスからの継承については理解できても，独自のクラスを構想し作成することは難しい - まあ，VC++ の学習が (体系的でなく) 付け焼き刃だからと言われればそれまでだが．これまでは，適当な参考書にしたがってあるプロジェクトを実現するようなコードを 1000 行程度書き移すというような学習で，BASIC や C コンパイラの使用ができるようになったのだが，Windows になってからはそれもままならなかった．

幸いにも 2006 年頃に，参考書 [1] を見て，VC++ 6.0 で LatticePCounter を Windows アプリケーションとして実装することができた．参考書 [1] を見るまでは，Windows 上での Win32_Console_Applications の作り方さえもわからなかった．

さて，LatticePCounter を，Visual C++ 2008 で Windows アプリケーションとして書き直そうとして，躓いてしまった - Generic クラスの使用法が (私のような素人には) わからなかったからである．MFC とは無関係に Generic クラスがあるのだが，この基礎的対象である Generic クラスについての VC++ のマニュアル，説明本や FAQ を尋ねても，作成方法や具体的操作が（どこかに説明されているのか？）わからない．

そういうことで，プロジェクト「LatticePCounter」において，プログラムを作成する部分，特に新しいクラスの追加について，その具体的手順を探さねばならなかった．

さて，Visual C++ について [http://msdn.microsoft.com/ja-jp/vstudio/hh220635.aspx] には，以下の様に解説されている．

VC++ は，MS-DOS 用 C/C++ コンパイラを起源に，Windows プラットフォーム向けの開発言語として Windows プラットフォームの成長と共に 40 年近くの歳月をかけて連綿と拡張が続けられたプログラミング言語です．

VC++ の使い方は大別すると，以下の 3 つに分類できます．

- Win32/COM API，MFC/ATL を使ったネイティブ プログラム開発用
- .NET Framework を使ったマネージ プログラム開発用
- 標準 C/C++ プログラム開発用

1 番目は，Windows が提供する機能をフル活用したプログラムを開発する手段としての VC++ です．ネイティブ プログラムとは，一般的な C/C++ がそうであるように，作成したプログラムが CPU 上で直接実行可能なバイナリ コードにコンパイルされ，高速実行されるプログラム形式です．Windows の諸機能はネイティブ プログラミングで利用することを前提とした Win32 API(Application Programming Interface) や COM (Component Object Model) のライブラリを通じて提供されます．ネイティブ プログラムで開発すると，Windows の機能をフルに使った高速なアプリケーションを開発可能です．また，ユーザー独自機能を COM で作成する場合には ATL (Active Template Library) を利用し，GUI(Graphical User Interface) を活用したアプリケーションを作成するには MFC (Microsoft Foundation Class) を利用します．

3 番目は，標準 C/C++ のプログラムを開発する手段としての VC++ です．この使い方に該当する代表的なケースは以下の通りです．UNIX 系で動いていたプログラムを Windows に移植する標準 C/C++ への準拠が開発要件として決められている Windows 以外の OS で動作するプログラムの開発環境として使う．

標準 C/C++ のプログラムを開発する場合，VC++ の言語仕様が ANSI や ISO 等で決められている C/C++ の標準にどの程度準拠しているかが問題になります．VC++ は，C 言語標準の ISO C90 に，C++ 言語標準の C++98 に対応し，それらに対応した C 言語の標準ライブラリや C++ の STL(Standard Template Library) も用意されています．

([http://msdn.microsoft.com/ja-jp/vstudio/hh220635.aspx] から引用した．)

参考書 [2] では，Visual C++ で作成されたプログラムについて，つぎのように指摘している：
作成されたプログラムの実行は C プログラムと違って，メッセージ+メッセージハンドラ関数 という関係で行われる．··· C 言語のような手続き型ではない．Visual C++ のプログラムは，

メッセージがこないかいつも監視している (メッセージテーブルを見ている)

メッセージがきたら指定されたメッセージ関数を実行する

という方法で実行される．たとえばユーザがマウスボタンをクリックすると，対応するメッセージ (WM_LBUTTONDOWN) が発行される．すると Visual C++ プログラムはメッセージテーブルを見て，マウスクリックに対応するメッセージハンドラ関数 (たとえば OnLBUTTONDOWN()) があれば，それを起動する．

Visual C++ プログラムの実行時間のほとんどは「メッセージテーブルを監視している」ということになる．

Visual C++ プログラムを作るということは，おおざっぱにいえば，

メッセージとメッセージハンドラ関数の対応を設定する

メッセージハンドラ関数を記述する

ということの繰り返しになるといわれる．

Visual C++ の特徴は以下のようにも言い表される：

- シングルドキュメントインターフェース (SDI)，マルチドキュメントインターフェース (MDI)，ダイアロベースのアプリケーション (Dialog Based Application) の原型が簡単に作成できる
- プログラムの骨格 (スケルトン) の作成を行う機能がある (AppWizard)
- 作成したスケルトンには基本的なメニューとツールバーが最初から用意されている
- クラスの追加・管理を行う機能がある (ClassWizard)
- プログラムのドキュメントとその処理部分を別管理してくれる (ドキュメントクラス)
- 固定的なデータはリソースとしてプログラム本体と分割管理する
- デバッグ機能が充実している
- Windows 上で運用するための有用な機能が豊富に用意されている
  (SDK: Software Development Kit などをいう？)
- ネイティブ形式のプログラムと，マネージ形式のプログラムを作ることができる
- アプリケーションを作成する方法もいくつか用意されており，ユーザーの使いやすい環境で開発することが可能であるその方法として次のようなものがある：
  - Windows API を使用する方法
  - MFC ライブラリを使用する方法

– ATL テンプレートライブラリを使用する方法
– Microsoft.NET Framework を利用し，C++/CLI を使用する方法． (CLI: Common Language Infrastructure)

Visual C++ のユーザーインターフェイスは極めて扱いやすい．したがって，この開発ツールを利用すれば，プログラムの基本的な構成を自分で記述する必要がなくなるため，手軽にプログラムを作成することが可能となる．つまり，プログラミングそのものではなく，プログラムの組立作業の効率化を追及したものとなっているといわれる．

このようなことを読み Web 上を彷徨いながら，ついに新しいクラスを定義しているヘッダをどこかで (Linker に?) 教えておく必要があるということに気づいた．新しいクラスを定義しているヘッダを どこかに include しておくことが必要だということ．ただそれだけで，コンパイルはできるがリンクは ”error C2143: 構文エラー” がでるというような問題を避けられるのだということに気づいた．

Visual C++ 2008 で MFC を使用することによってアプリケーションを作成することにした．

Visual C++ 2008 の Windows アプリケーションについて

Visual C++ 2008 の MFC を利用して作成されるアプリケーションの内で，ひとつのドキュメントを利用する SDI 形式のアプリケーションとして『LatticePCounter』を作成する．

参考書 [1] によれば，ドキュメント/ビュー アーキテクチャー (Document/View Architecture) においては，

- CDocument クラスは，CView クラスから入力を受け取ったり，表示する情報を CView クラスに渡したりする．さらに，CDocument クラスでは，ドキュメントデータの保存やファイルなどからの復元も行うという．
- CView クラスは，ユーザに対してドキュメントを表示する役割を持つ．CView クラスは，入力情報を CDocument クラスに渡したり，表示する情報を CDocument クラスから受け取ったりするという．このクラスに追加するコードの大半は，ドキュメントの表示や入力処理に関するもので占められるという．

簡単に，ドキュメント (Document) はアプリケーションを使って作成したデータを扱うためのもの，ビュー (View) がデータの表示を提供するものと理解できる．

こうして， LatticePCounter の **仕様　格子多角形の取得** については：

(マウスの左クリックで多角図形の頂点 $P_1, P_2, \cdots, P_{n-1}$ を拾った後，) マウスの右クリックで頂点 P_n を拾ったというメッセージ OnRButtonUp を受けて，対応するメッセージハンドラ関数

void CLatticePCounterView::OnRButtonUp(UINT nFlags, CPoint point)

内に，辺周上の格子点をプロットし数えた上で多角形であるかどうか判定するコードを記述することになる．

仕様　格子多角形の内点の取得 については：

すでにメッセージ OnRButtonUp を受けているから，メッセージハンドラ関数

void CLatticePCounterView::OnRButtonUp(UINT nFlags, CPoint point)

内に，多角図形が多角形であるときに内部の格子点となり得る格子点の位置を取得し数え上げつつ，これらの格子点をプロットするコードを追加記述することになる．

これらの仕様を実現するために，多角図形の Horn を多角図形に伴う構造体 horn として考える．

1. **仕様　格子多角形の取得** のコードを記述においては， マウスで取り出した多角図形 (辺の集合) P : $P_1P_2, P_2P_3, \cdots, P_nP_1$ が多角形であるかどうか判定する．

   マウスで取り出した多角図形 P が”多角形でない”ことのエラーチェックは

   - 多角図形 P の頂点の二重頂点の有無については，関数

     ```
     int check_double( int , CPoint [])
     ```

     で調べ，有れば二重頂点の除去修正も行う．
   - 多角図形 P の Elongated Horn(退化角) の有無については，関数

     ```
     void check_orientation( int n, int *h, CPoint p[])
     ```

     で調べ， Elongated Horn に伴う Elongated-Horn Error の情報を得る．この関数は多角図形 P の頂点の向き付けを調べ，その向き付けの正を確認または向き付けを正に設定する．
   - 多角図形 P の辺周上の総格子点数 nbp の取得については，関数

     ```
     long integral_boundary( int , CPoint [] , CPoint [])
     ```

     の返り値によって知る．このとき，多角形の周上の格子点をすべて多角形の頂点化しておく．
   - 多角図形 P の頂点の重畳の有無については，関数

     ```
     void check_overlap( int n, CPoint p[])
     ```

     によって知り，Overlapping-Error の情報を得る．
   - 多角図形 P の 辺の交差状態ついては，関数

     ```
     void check_intersect( int n, CPoint p[])
     ```

     によって，Intersect Error の情報を得る．

   この様にして，多角図形 P が多角形であるかどうか判定する．そうではあるが，多角図形 P が”多角形でない”ことの判定については **仕様　格子多角形の内点の取得** のコード中の関数

   ```
   void getatarget( struct horn *pc)
   ```

   が与える構造体 horn に関する情報 Winding Error で二重にチェックしている．

2. **仕様　格子多角形の内点の取得** のコードを記述においては，多角図形の辺周上の格子点がすべて多角図形の頂点となっている場合を考えている．

- 多角図形の Horn 情報については，関数

  ```
  void getatarget( struct horn *)
  ```

  によって，定理１．にいう多角形の Horn 情報を得る．

- 多角図形に含まれている格子点の取得については，関数

  ```
  void searchvx( int , struct horn *, CPoint [] , CPoint *)
  ```

  によって Horn 情報から（多角図形が多角形である場合にこの多角形に含まれている）格子点 newp を取得する．

- 多角図形の内部の格子点数 nip を取得については，関数

  ```
  int putvx( int , int , int [] , struct horn *, CPoint [] , CPoint [] , CPoint *)
  ```

  （多角形の辺周上の格子点がすべて多角形の頂点であるとき！）を通して，この格子点 newp が辺周上の点か内部の点かを判断して多角図形の内部の格子点数 nip を取得する．

  さらに，

- 格子点 newp から新たな多角図形を構成し還元処理する．
  途中，二つの多角図形へ分解が生じ得るので，関数

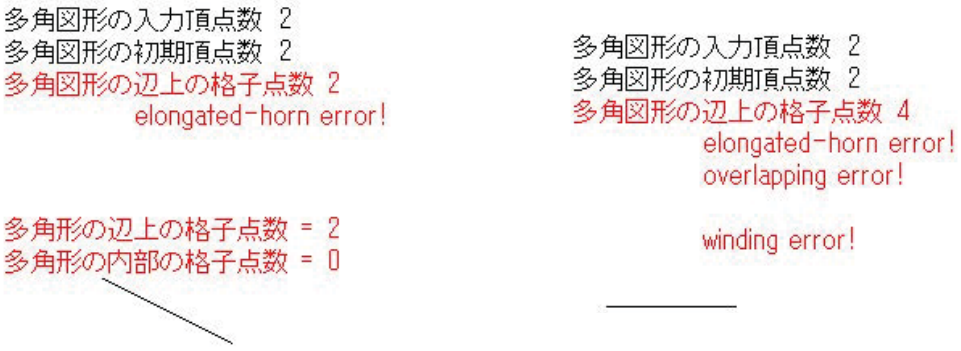
  ```
  int subpolygon( int , int , int , int [] , CPoint [] , CPoint [] );
  ```

  で部分多角図形を取得しフラグ (int) suc を立て，各多角図形を逐次処理する．

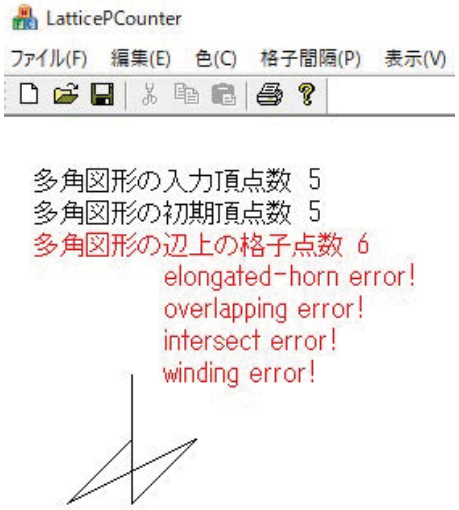
- 多角形の内部の格子点数は，この手続きが Winding Error なしに完了したとき，多角図形の内部の格子点数 nip として得られる．

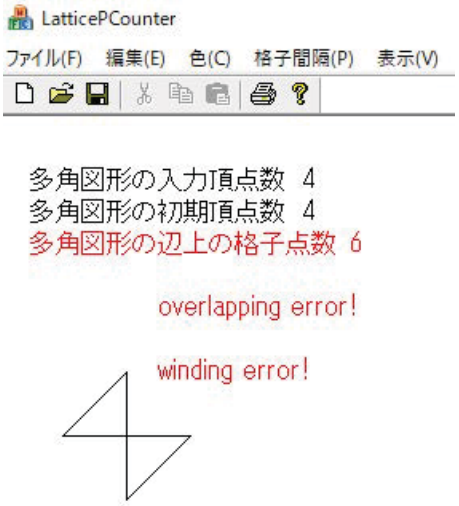
- Winding Error の前に得られた多角図形の内部の格子点の描画を行う．

このアプリケーション『LatticePCounter』では，多角形ではない多角図形にたいしても**仕様　格子多角形の内点の取得**に関わる関数を実行させている；　このことから多角図形の他の情報が得られると思う．例えば，下図の線分を見よ．

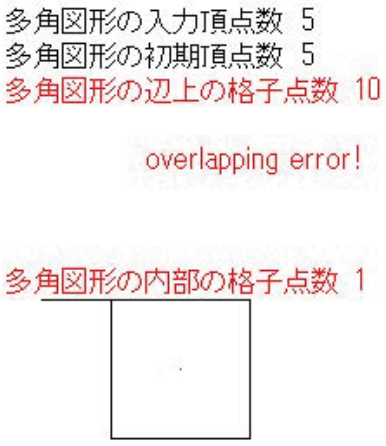


例えば，下図の場合には， 巻き方の誤り winding error! と 格子点の重畳 overlapping error! が共通している (多角図形は多角形でない) が，エラーに伴って多角図形の辺上の格子点数を特徴的に誤っている．多角図形の辺上の格子点数は相異なる格子点数ではないからである．





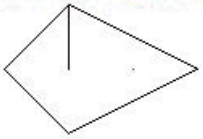
下図では，格子点の重畳 overlapping error! が共通しているので二つの多角図形は多角形でないが，正方形の中に格子点はプロットされている．巻き方の誤り winding error! を取られていない右図の場合に表示されている「多角図形内部の格子点数」とは，プロットした（多角形内部の格子点の可能性があった）格子点の数を参照データとして表示しているものである．



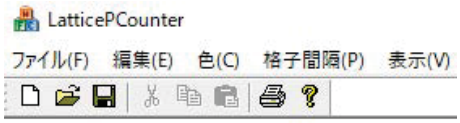
多角図形の入力頂点数 6
多角図形の初期頂点数 6
多角図形の辺上の格子点数 6

overlapping error!

多角図形の内部の格子点数 1



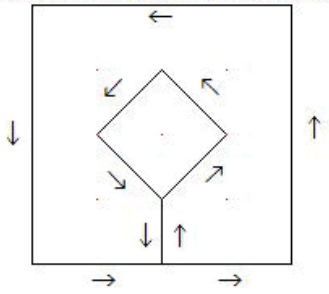
つぎのような場合もある．多角図形の正の向きは多角図形の Horn で決められているが，多角図形によっては必ずしも一意的には決まらない．その多角図形の正の向き付けの違いが，下図の場合のような差異をもたらすこともある．



多角図形の入力頂点数 11
多角図形の初期頂点数 11
多角図形の辺上の格子点数 22

overlapping error!

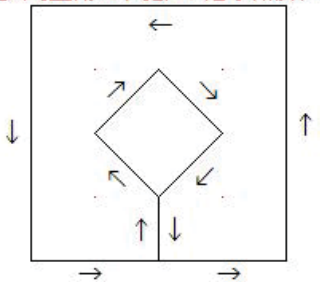
多角図形の内部の格子点数 8



多角図形の入力頂点数 11
多角図形の初期頂点数 11
多角図形の辺上の格子点数 22

overlapping error!

多角図形の内部の格子点数 4



このアルゴリズムについては，二つの考察すべき課題： 多角形内の格子点の位置を取得し数え上げる手続きの性能評価と計算量評価 が残るが，今は放置する．

3　格子多角形の取得に関する仕様の実装

Windows アプリケーション LatticePCounter に格子多角形のデータ取得と描画の機能を与える．

シングルドキュメント形式 (SDI) のアプリケーションの作成手順

多角形データの判定と取得の実装

1. Windows のプログラムメニューから [Microsoft Visual Studio 2008] を選択し，[Microsoft Visual Studio 2008] を起動する．
2. メニューの [ファイル (F)] − [新規作成 (N)] − [プロジェクト (P)] を選択する．
3. 新しいプロジェクトウィンドウの [プロジェクトの種類 (P)] を [MFC] − [テンプレートの種類 (T)] を [MFC アプリケーション] と選択する．
4. [プロジェクト名 (N)] に適切な名前を入力した後 [OK] ボタンをクリックする．ここでは，プロジェクト名を「LatticePCounter」とした．
5.「概要」画面での設定この後詳細設定を行うため，ここではなにも操作は行わずに [次へ] をクリックする．
6.「アプリケーションの種類」画面での設定 [シングルドキュメント (S)] を選択し，[ユニコードライブラリを使用する (N)] のチェックをはずし [次へ] をクリックする．
7.「複合ドキュメントサポート」画面での設定デフォルトの [なし (N)] のまま [次へ] をクリックする．
8.「ドキュメントテンプレート文字列」画面での設定デフォルトのまま設定のまま [次へ] をクリックする．
9.「データベースサポート」画面での設定デフォルトの [なし (N)] のまま [次へ] をクリックする．
10.「ユーザーインターフェイス」画面での設定ここではユーザーの使いやすい環境に設定するのだが，デフォルトの設定で問題ないので，何も操作をせず [次へ] をクリックする．(ユーザーに合わせて設定を変えてもよい．
11.「高度な機能の設定」画面での設定 [ActiveX コントロール (R)] のチェックをはずして [次へ] をクリック．
12.「生成されたクラス」画面での設定基本クラス (A) を [CView] クラスに設定し，[完了] ボタンをクリックする．
13. プログラムを作成する．
14. 作成したプログラムのエラーチェックを行う．メニューの [ビルド (B)] − [LatticePCounter のビルド (U)] を選択する．
15. エラーが出た場合は，エラーを修正し再度コンパイルを行う．メニューの [ビルド (B)] − [LatticeP-Counter のリビルド (E)] を選択する．
16. 14 − 15 の手順を繰り返し，エラーが取り除けたらプログラムを実行する．メニューの [デバッグ (D)] − [デバッグ開始 (S)] を選択する．もしくはツールバーにある緑色の▲ボタンをクリックする．

以上のような手順でアプリケーションを作成することができる．手順 13「プログラムを作成する．」の部分でどのようなコードを追加するかによって，どのようなアプリケーションになるのかが決まる．例えば，グラフを書いたり図形を描画するだけならば，OnDraw 関数の中にコードを追加するだけでよい．

プログラム LatticePCounter では (二つの端点の組である) クラス CLine を新しいクラスとして生成する必要がある．そこで，プロジェクト「LatticePCounter」において，プログラムを作成する部分，特に新しい CLine クラスの追加から，その具体的手順を述べる．

プログラムを作成する．

格子多角形のデータ取得と描画

Windows アプリケーション として，マウスの左クリックで多角図形の頂点 $P_1, P_2, \cdots, P_{n-1}$ を拾い，右クリックで頂点 P_n を拾い頂点 P_1 へ結んで辺周として閉じて，多角図形 P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ とした後，つぎの情報を獲得する：

- マウスで取り出した多角図形 (辺の集合) P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ の辺周上の格子点を探し出し，辺周上の格子点をプロットし数える（初期情報）.
- マウスで取り出した多角図形 (辺周) P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ が多角形であるためのいくつかの基本的事項，例えば 頂点の重畳，辺の重畳，辺の交差等を調べて多角図形の頂点の情報を再取得する.
- 辺周上の格子点をプロットし，辺周上の格子点数を確認する.

詳しい手順は以下の通りである.

1. 新しいクラスの追加 (CLine クラス)
   直線のデータを扱うために CLine クラスを追加する.
   (a) メニューの [プロジェクト (P)] − [クラスの追加 (C)] を選択する.
   (b) [カテゴリ (C)] を [VisualC++] − [テンプレート (T)] を [C++ クラス] を選択し，[追加] ボタンをクリックする.
   (c) 汎用 C++ クラスウィザードにおいて，[クラスの名前] に [CLine] と入力．そして，[アクセス (A)] − [public]，[基本クラス] − [CObject] を選択し，[完了] ボタンをクリックする.

2. CLine クラスのインスタンスの初期化
   CLine クラスにメンバ変数 (m_ptFrom , m_ptTo) を追加する.
   (a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.
   (b) ツリーを開き，[CLine] クラスを右クリックし，メニューを開く.
   (c) メニューの中から [追加] − [変数の追加 (B)] を選択する.
   (d) メンバ変数の追加ウィザードにおいて，[アクセス (A)] − [private] を選択し，[変数の種類 (V)] に [CPoint]，[変数名 (N)] に [m_ptFrom] と入力し，[完了] ボタンをクリックする.
   (e) 同様にしてメンバ変数 [m_ptTo] を追加する.

   CLine クラスに CLine 関数を追加する.
   (a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.
   (b) ツリーを開き，[CLine] クラスを右クリックし，メニューを開く.
   (c) メニューの中から [追加] − [関数の追加 (U)] を選択する.

(d) メンバ関数の追加ウィザードにおいて，[戻り値の型 (Y)] には何も入力せず，[パラメータの型 (T)] に [CPoint]，[関数の名前 (U)] に [CLine] と入力する．[アクセス (E)] − [public] を選択する．
(e) [パラメータ名 (N)] に [ptFrom] と入力し，[追加 (A)] ボタンをクリックする．
(f) [パラメータ名 (N)] に [ptTo] と入力し，[追加 (A)] ボタンをクリックする．
(g) [パラメータの一覧 (L)] に [CPoint ptFrom] と [CPoint ptTo] が追加されていることを確認し，[完了] ボタンをクリックする．
(h) 作成した関数を編集し，Line.cpp にコードを追加する．

```
CLine::CLine(CPoint ptFrom, CPoint ptTo)
{
    // 直線の始点と終点の座標の値の初期化
    m_ptFrom = ptFrom ;
    m_ptTo = ptTo ;
}
```

3. CLine クラスに描画機能を加える．
CLine クラスに Draw 関数を追加する．
(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する．
(b) ツリーを開き，[CLine] クラスを右クリックし，メニューを開く．
(c) メニューの中から [追加] − [関数の追加 (U)] を選択する．
(d) [戻り値の型 (Y)] − [void] を選択し，[パラメータの型 (T)] に [CDC*]，[関数の名前 (U)] に [Draw] と入力する．[アクセス (E)] − [public] を選択する．
(e) [パラメータ名 (N)] に [pDC] と入力し，[追加 (A)] ボタンをクリックする．
(f) [パラメータの一覧 (L)] に [CDC* pDC] が追加されていることを確認し，[完了] ボタンをクリックする．
(g) (Line.cpp で) 作成した Draw 関数にコードを追加する．

```
void CLine::Draw(CDC* pDC)
{
    // 直線の描画
    pDC->MoveTo(m_ptFrom);
    pDC->LineTo(m_ptTo);
}
```

4. ドキュメントクラスに機能を追加する．
ドキュメントクラスにメンバ変数 (m_oaLines) を追加する．
(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する．
(b) ツリーを開き，[CLatticePCounterDoc] クラスを右クリックし，メニューを開く．
(c) メニューの中から [追加] − [変数の追加 (B)] を選択する．
(d) メンバ変数の追加ウィザードにおいて，[アクセス (A)] − [public] を選択し，[変数の種類 (V)] に [CObArray]，[変数名 (N)] − [m_oaLines] と入力し，[完了] ボタンをクリックする．

5. 線を追加する．

ドキュメントクラスに AddLine 関数を追加する．

(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する．

(b) ツリーを開き，[CLatticePCounterDoc] クラスを右クリックし，メニューを開く．

(c) メニューの中から [追加] − [関数の追加 (U)] を選択する．

(d) [戻り値の型 (Y)] に [CLine*]，[パラメータの型 (T)] に [CPoint]，[関数の名前 (U)] に [AddLine] と入力する．[アクセス (E)] − [public] を選択する．

(e) [パラメータ名 (N)] に [ptFrom] と入力し，[追加 (A)] ボタンをクリックする．

(f) [パラメータ名 (N)] に [ptTo] と入力し，[追加 (A)] ボタンをクリックする．

(g) [パラメータの一覧 (L)] に [CPoint ptFrom] と [CPoint ptTo] が追加されていることを確認し，[完了] ボタンをクリックする．

(h) (LatticePCounterDoc.cpp で) 作成した AddLine 関数を編集し，コードを追加する．

```
CLine* CLatticePCounterDoc::AddLine(CPoint ptFrom, CPoint ptTo)
{
    // 新しいCLineオブジェクトの生成
    CLine *pLine = new CLine(ptFrom, ptTo) ;
    try
    {
        // 新しいCLineオブジェクトをオブジェクトの配列に追加
        m_oaLines.Add(pLine) ;
        // ドキュメントに変更が加えられたことを通知するフラグをONにする
        SetModifiedFlag() ;
    }
    // メモリが許容範囲を超えた場合は例外処理を行う
    catch(CMemoryException* perr)
    {
        // 警告メッセージの表示
        AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK) ;
        // CLineオブジェクトを作成したかどうか
        if(pLine)
        {
            // CLineオブジェクトを解放
            delete pLine ;
            pLine = NULL ;
        }
        // 例外オブジェクトを解放
        perr->Delete() ;
    }
```

```
    return pLine;
}
```

6. 直線の数を取得する．
   ドキュメントクラスに GetLineCount 関数を追加する．
   (a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する．
   (b) ツリーを開き，[CLatticePCounterDoc] クラスを右クリックし，メニューを開く．
   (c) メニューの中から [追加] − [関数の追加 (U)] を選択する．
   (d) [戻り値の型 (Y)] − [int] を選択し，[パラメータの型 (T)] には何も入力しない．
   (e) [関数の名前 (U)] に [GetLineCount] と入力し，[アクセス (E)] − [public] を選択する．
   (f) [パラメータ名 (N)] には何も入力せず，[完了] ボタンをクリックする．
   (g) (LatticePCounterDoc.cpp で) 作成した GetLineCount 関数を編集し，コードを追加する．

```
int CLatticePCounterDoc::GetLineCount(void)
{
    // 配列のカウントを返す
    return (int) m_oaLines.GetSize();
}
```

7. 直線のオブジェクトを取得する．
   ドキュメントクラスに GetLine 関数を追加する．
   (a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する
   (b) ツリーを開き，[CLatticePCounterDoc] クラスを右クリックし，メニューを開く．
   (c) メニューの中から [追加] − [関数の追加 (U)] を選択する．
   (d) [戻り値の型 (Y)] に [CLine*] と入力し，[パラメータの型 (T)] − [int] を選択する．
   (e) [関数の名前 (U)] に [GetLine] と入力し，[アクセス (E)] − [public] を選択する．
   (f) [パラメータ名 (N)] に [nIndex] と入力し，[追加 (A)] ボタンをクリックする．
   (g) [パラメータの一覧 (L)] に [int nIndex] が追加されていることを確認し，[完了] ボタンをクリックする．
   (h) (LatticePCounterDoc.cpp で) 作成した GetLine 関数にコードを追加する．

```
CLine* CLatticePCounterDoc::GetLine(int nIndex)
{
    // オブジェクト配列中のnIndex番目のCLineオブジェクトへのポインタを返す
    return (CLine*)m_oaLines[nIndex];
}
```

8. LatticePCounterDoc.h に ♯include ”Line.h”.
   LatticePCounterDoc.h の中で ♯pragma once の後に，♯include ”Line.h” を追加しておく．

   注意： または，CLine を定義しているヘッダを include するディレクトリを指定しておく

(a) プロジェクトのプロパティ→構成プロパティ→ C/C++ →全般→追加のインクルードディレクトリに Line.h が入っているディレクトリを指定する.

(b) インクルードディレクトリの設定 [ツール] → [オプション] → [プロジェクトおよびソリューション] → [VC++ ディレクトリ] → [ディレクトリを表示するプロジェクト] → [インクルード ファイル] に記述しておく！

9. 描画データを表示する.

ビュークラスにメンバ変数 (m_ptPrevPos) を追加する.

(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.

(b) ツリーを開き，[CLatticePCounterView] クラスを右クリックし，メニューを開く.

(c) メニューの中から [追加] − [変数の追加 (B)] を選択する.

(d) メンバ変数の追加ウィザードにおいて，[アクセス (A)] − [private] を選択し，[変数の種類 (V)] に [CPoint]，[変数名 (N)] に [m_ptPrevPos] と入力し，[完了] ボタンをクリックする.

10. マウスイベントを追加する.

ビュークラスにメッセージハンドラ関数を追加する.

(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.

(b) ツリーを開き，[CLatticePCounterView] クラスを右クリックし，[プロパティ (R)] を選択する.

(c) プロパティ画面の [メッセージタブ] を選択し，メッセージハンドラの一覧を表示する. [WM_LBUTTONDOWN][WM_RBUTTONDOWN][WM_RBUTTONUP] の 3 つのメッセージハンドラを探し出し，それぞれ追加する.

(d) (LatticePCounterView.cpp で) 追加した 3 つの関数のコードを編集する.

```
void CLatticePCounterView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を
    呼び出します。

    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    //  マウスをキャプチャしているかどうかを確認
        if (GetCapture() == this)
    {
    //  デバイスコンテキスタを取得する
        CClientDC dc(this);

        mod_vertex( point, &vertex, &coverp);
    //  ドキュメントにCLineオブジェクトを追加
```

```
            CLine *pLine = GetDocument() -> AddLine(m_ptPrevPos, coverp);

//  入力された直線を描画
    pLine -> Draw(&dc);

//  直前の座標として現在の座標を取得
    m_ptPrevPos = coverp;
    p[vertexN] = vertex;
    vertexN++;
} else
{
    //  Document を更新
    CLatticePCounterDoc* pDoc = GetDocument();
    pDoc -> DeleteContents();

    //  View を更新
    Invalidate();

    vertexN=1;
    initpolygon( p, q, nps, pit);

    // マウスの入力を、カーソルの位置にかかわらず全て受け取り、
    // 他のウィンドウに取得されないようにする
    SetCapture();

    mod_vertex( point, &vertex, &coverp);

    ptA = coverp;

    //  直前の座標として現在の座標を取得
    m_ptPrevPos = coverp;    // m_ptPrevPos = point;
    p[vertexN] = vertex;     // p[vertexN] = point;
    vertexN++;
    }

//////////////////////////
// コードの終わり
```

```
    ////////////////////////

    CView::OnLButtonDown(nFlags, point);
}

void CLatticePCounterView::OnRButtonDown(UINT nFlags, CPoint point)
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を
    呼び出します。

    ////////////////////////
    // コードの始まり
    ////////////////////////

    //  マウスをキャプチャしているかどうかを確認
    if (GetCapture() == this) {

    //  デバイスコンテキスタを取得する
        CClientDC dc(this);

        mod_vertex( point, &vertex, &coverp);
    //  ドキュメントにCLineオブジェクトを追加
        CLine *pLine = GetDocument() -> AddLine(m_ptPrevPos, coverp);

    //  入力された直線を描画
        pLine -> Draw(&dc);

    //  直前の座標として現在の座標を取得
        m_ptPrevPos = coverp;
        p[vertexN] = vertex;

        vertexT = vertexN;
        lineN = vertexT;

    //  多角形の入力頂点数を確認
        char buffer[20];
        _itoa_s( vertexT, buffer, 10 );
```

```
        CString Choten= buffer;
        dc.TextOut(20, 30, "多角図形の入力頂点数  ");
        dc.TextOut(190, 30, Choten);
        };

    ////////////////////////
    // コードの終わり
    ////////////////////////

    CView::OnRButtonDown(nFlags, point);
}

void CLatticePCounterView::OnRButtonUp(UINT nFlags, CPoint point)
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を
    呼び出します。

    ////////////////////////
    // コードの始まり
    ////////////////////////

    //  マウスをキャプチャしているかどうかを確認
    if (GetCapture() == this) {

    //  デバイスコンテキスタを取得する
        CClientDC dc(this);

    //  ドキュメントにCLineオブジェクトを追加
        CLine *pLine = GetDocument() -> AddLine(m_ptPrevPos, ptA);

    //  入力された直線を描画
        pLine -> Draw(&dc);      //  入力された直線の数=入力頂点数


        char buffer[20];
        CString Choten= buffer;
```

```
//  多角形の初期頂点数
    np = vertexN;
    endpoint(np, p);
    np = check_double(np, p);       //    二重頂点の除去

    _itoa_s( np, buffer, 10 );
    Choten= buffer;
    dc.TextOut(20, 50, "多角図形の初期頂点数  ");
    dc.TextOut(190, 50, Choten); // 初期頂点数の TextOut

//  多角形の向き付けの確認と Elongated Horn の有無
    check_orientation(np, &hx, p);
    if(!nline) {  SetTextColor(dc, RGB(255, 0, 0));    // Red
                  dc.TextOut(100, 90, "elongated-horn error!");
    }

//  多角形の周上の格子点数 nbp の取得
    nbp=integral_boundary(np, p, q);
    np = nbp;    //    多角形の周上の格子点はすべて多角形の頂点
    endpoint(np, p);

//  多角形の頂点の重畳の有無の確認
    check_overlap(np, p);
    if(overlap!=-1) { SetTextColor(dc, RGB(255, 0, 0)); // Red
                      dc.TextOut(100, 110, "overlapping-error!");
    }

//  多角形の周上の格子点数 nbp の TextOut
    _itoa_s( nbp, buffer, 10 );
    Choten= buffer;
    dc.TextOut(20, 70, "多角図形の辺上の格子点数  ");
    dc.TextOut(220, 70, Choten);

//  多角形の辺の自己交差の確認
    check_intersect( np, p);
    if (intersect!=-1) { SetTextColor(dc, RGB(255, 0, 0));
```

```
                                dc.TextOut(100, 130, "intersect error!");
        };

        // マウスをキャプチャしていた場合は、他のウインドウが
        // マウスの入力を取得できるようにマウスをリリースする
        ReleaseCapture();
    };

    ////////////////////////
    // コードの終わり
    ////////////////////////

    CView::OnRButtonUp(nFlags, point);
}
```

11. 描画を行う.
    (a) CLatticePCounterView クラスの OnDraw 関数にドキュメントのデータを描画するコードを追加する.

```
void CLatticePCounterView::OnDraw(CDC* pDC)
{
    CLatticePCounterDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: この場所にネイティブ データ用の描画コードを追加します。

    ////////////////////////
    // コードの始まり
    ////////////////////////

    // ドキュメント内の直線の数を取得する
    int liCount = pDoc -> GetLineCount();

    // ドキュメント内に直線が無い場合は処理を行わない
    if (liCount)
    {
```

```
        int liPos;
        CLine *lptLine;

        // 直線の数だけループ
        for (liPos = 0; liPos < liCount; liPos++)
        {
            // ドキュメント内に含まれる CLINE オブジェクトを順番に取得する
            lptLine = pDoc -> GetLine(liPos);

            // 取得した CLINE オブジェクトを描画
            lptLine -> Draw(pDC);
        }
    }


    ////////////////////////
    // コードの終わり
    ////////////////////////
}
```

(b) CLine クラスのヘッダーファイルを，ドキュメントクラスとビュークラスのソースファイル (LatticePCounterDoc.cpp と LatticePCounterView.cpp) にインクルードする：

```
♯ include "stdafx.h"
♯ include "LatticePCounter.h"
♯ include "MainFrm.h"
♯ include "Line.h"
♯ include "LatticePCounterDoc.h"
♯ include "LatticePCounterView.h" .
```

(c) (外部) 関数等を，ビュークラスのソースファイル LatticePCounterView.cpp に記述する.

```
#define EPOINTS 25600    //2560
#define INPOINTS 102400    //10240
#define HPOINTS 500    //50


struct horn{
                int x0, y0;       // x0 = xl < 0 ,  y0 = yl
                int x1, y1;       // x1 = xr >= 0 ,  y1 = yr
```

```
                int cx, cy;      // xr*cy - yr*cx = gcd(xr, yr)
               long px, py;      // px = -cy*xl + cx*yl ,
                                 // py = yr*xl - xr*yl > 0
} HORN = { 0, 0, 0, 0, 0, 0, 0l, 0l}, *pc=&HORN;

CPoint ptA, m_ptPrevPos;
CPoint p[EPOINTS]; //CPoint p[2560];
CPoint q[EPOINTS]; //CPoint q[2560];
CPoint inp[INPOINTS]; //CPoint inp[10240];

int nps[HPOINTS]={0}; //int nps[50]={0};
int vertexN=0, vertexT=0, lineN=0;
int pit=10, suc, np, hx, pk, fn, flg;    // nsgn 向き付け
int nline,  intersect=-1,  overlap=-1,  polygoncode=-1;
long nbp;

CPoint newp=(100, 100), *w=&newp;
CPoint vertex=(100, 100), *v=&vertex;
CPoint coverp=(100, 100), *c=&coverp;
CPoint frP=(100, 100), *f=&frP;
CPoint toP=(100, 100), *t=&toP;

int gcd( int, int);
int euclid( CPoint *);

void initpolygon( CPoint [], CPoint [], int [], int);
void mod_vertex( CPoint, CPoint *, CPoint *);
void endpoint( int, CPoint []);
int delvxs( int, int, int, CPoint []);
int check_double( int, CPoint []);
void high_light( int, int *, CPoint []);
void both_sides( int , struct horn *, CPoint []);
void check_orientation( int, int *, CPoint []);
void check_overlap( int, CPoint []);
void check_intersect( int, CPoint []);
int line_intersect( CPoint,  CPoint,  CPoint,  CPoint);
long integral_boundary( int, CPoint [], CPoint []);
```

```
int check_w( int , int , CPoint [] , CPoint *);
```

(d) (外部) 関数のコードをビュークラスのソースファイル LatticePCounterView.cpp に記述する：
4 (外部) 関数のコード を参照.

12. グラフィックスを削除する.
ドキュメントクラスに DeleteContents 関数を追加する.
(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.
(b) ツリーを開き，[CLatticePCounterDoc] クラスを右クリックし，メニューを開く.
(c) メニューの中から [追加] − [関数の追加 (U)] を選択する.
(d) [戻り値の型 (Y)] に [void] と入力し，[パラメータの型 (T)] − [] を選択する.
(e) [関数の名前 (U)] に [DeleteContents] と入力し，[アクセス (E)] − [public] を選択する.
(f) [完了] ボタンをクリックし，(LatticePCounterDoc.cpp で) 作成した DeleteContents 関数にコードを追加する.

```
void CLatticePCounterDoc::DeleteContents(void)
{
    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    // オブジェクト配列に含まれる CLine オブジェクトの数を取得
    int liCount = (int) m_oaLines.GetSize();    // CLineオブジェクトの数
    int liPos;                                  // 任意のCLineオブジェクトの数

    // オブジェクト配列に１つでも CLine オブジェクトがあるか判定
    if (liCount)
    {
        // 配列内をループし、CLine オブジェクトをすべて消去する
        for (liPos = 0; liPos < liCount; liPos++)
        delete m_oaLines[liPos];
        // 配列内をリセットする
        m_oaLines.RemoveAll();
    }

    ////////////////////////////
    // コードの終わり
    ////////////////////////////

    CDocument::DeleteContents();
```

```
}
```

13. データの保存とファイルからの読み込みを行う.
ドキュメントクラス内の Serialize 関数内のコードの編集を行う (LatticePCounterDoc.cpp で)：

```
// CLatticePCounterDoc シリアル化
void CLatticePCounterDoc::Serialize(CArchive& ar)
{
        // オブジェクト配列をシリアル化する
        m_oaLines.Serialize(ar);
}
```

Line クラスに Serialize 関数を追加する.
(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.
(b) ツリーを開き，[CLine] クラスを右クリックし，メニューを開く.
(c) メニューの中から [追加] − [関数の追加 (U)] を選択する.
(d) [戻り値の型 (Y)] − [void] を選択し，[パラメータの型 (T)] には [CArchive&] と入力し,[アクセス (A)] − [public] を選択する.
(e) [パラメータ名 (N)] には [ar] と入力し [virtual] をチェックする.
(f) [関数の名前 (U)] に [Serialize] と入力し，[追加 (A)] ボタンをクリックする.
(g) [パラメータの一覧 (L)] に [CArchive& ar] が追加されていることを確認し，[完了] ボタンをクリックする.
(h) (Line.cpp で) コードを追加する.

```
void CLine::Serialize(CArchive& ar)
{
    CObject::Serialize( ar);

    // 色の選択
    if (ar.IsStoring())
        ar << m_ptFrom << m_ptTo ;
    else
        ar >> m_ptFrom >> m_ptTo ;
}
```

Line.h に“ DECLARE_SERIAL (CLine) ” を追加し，Line.cpp に“ IMPLEMENT_SERIAL (CLine, CObject, 1) ” を追加する.

14. Cline クラスに色の情報を追加する.
CLine クラスにメンバ変数　m_crColor を追加する.
(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.

(b) ツリーを開き，[CLine] クラスを右クリックし，メニューを開く．

(c) メニューの中から [追加] − [変数の追加 (B)] を選択する．

(d) メンバ変数の追加ウィザードにおいて，[アクセス (A)] − [private] を選択し，[変数の種類 (V)] に [COLORREF]，[変数名 (N)] に [m_crColor] と入力し，[完了] ボタンをクリックする．

(e) [クラスビュー (A)] の [CLine] コンストラクタ上で右クリックし，ポップアップメニューから宣言へ移動 (A) を選択する．

(f) [Cline] コンストラクタの宣言に第三引数として，[COLORREF crColor] を追加する．Line.h で

```
class CLine :
        public CObject
{
        DECLARE_SERIAL (CLine)
public:
        CLine(void);
        ~CLine(void);
private:
        CPoint m_ptFrom;
        CPoint m_ptTo;
public:
        CLine(CPoint ptFrom, CPoint ptTo, COLORREF crColor);
        void Draw(CDC* pDC);
        virtual void Serialize(CArchive& ar);
private:
        COLORREF m_crColor;
};
```

(g) [FileView] から，Line.cpp を開き編集し，[Cline] コンストラクタのコードを表示する．
Line.cpp で m_crColor メンバ変数を初期化するコードも追加する．

```
CLine::CLine(CPoint ptFrom, CPoint ptTo, COLORREF crColor)
{
    // 直線の始点と終点の座標の値の初期化
    m_ptFrom = ptFrom ;
    m_ptTo = ptTo ;
    m_crColor = crColor ;
}
```

(h) Line.cpp 中の Draw 関数も修正追加する．

```
void CLine::Draw(CDC* pDC)
{
    // ペンの作成
```

```
        CPen lpen (PS_SOLID, 1, m_crColor);

        // 新しく作成したペンを選択
        CPen* poldPen = pDC -> SelectObject(&lpen);

        // 直線の描画
        pDC->MoveTo(m_ptFrom);
        pDC->LineTo(m_ptTo);

        // 元のペンに戻す
        pDC -> SelectObject(poldPen);
    }
```

(i) Line.cpp 中の Serialize 関数も修正追加する.

```
void CLine::Serialize(CArchive& ar)
{
        CObject::Serialize( ar);

        // 色の選択
        if (ar.IsStoring())
                ar << m_ptFrom << m_ptTo << (DWORD) m_crColor ;
        else
                ar >> m_ptFrom >> m_ptTo >> (DWORD) m_crColor ;
}
```

15. CLatticePCounterDoc クラスに色の情報を追加する.

CLatticePCounterDoc クラスにメンバ変数 m_nColor を追加する.

(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.
(b) ツリーを開き, [CLatticePCounterDoc] クラスを右クリックし, メニューを開く.
(c) メニューの中から [追加] − [変数の追加 (B)] を選択する.
(d) メンバ変数の追加ウィザードにおいて, [アクセス (A)] − [private] を選択し, [変数の種類 (V)] に [UINT], [変数名 (N)] に [m_nColor] と入力し, [完了] ボタンをクリックする.

CLatticePCounterDoc クラスにメンバ変数 m_crColors[8] を追加する.

(a) メニューの [表示 (V)] − [クラスビュー (A)] を選択する.
(b) ツリーを開き, [CLatticePCounterDoc] クラスを右クリックし, メニューを開く.
(c) メニューの中から [追加] − [変数の追加 (B)] を選択する.
(d) メンバ変数の追加ウィザードにおいて, [アクセス (A)] − [public] を選択し, [変数の種類 (V)] に [static const COLORREF[8]], [変数名 (N)] に [m_crColors] と入力し, [完了] ボタンをクリック

する．

(e) LatticePCounterDoc.cpp を開き，m_crColors の色のテーブルを追加する．

```
  ⋮

END_MESSAGE_MAP()
```

の後ろに

```
const COLORREF CLatticePCounterDoc::m_crColors[8] ={
        RGB(  0,  0,  0),     // 黒
        RGB(  0,  0,255),     // 青
        RGB(  0,255,  0),     // 緑
        RGB(  0,255,255),     // 青緑
        RGB(255,  0,  0),     // 赤
        RGB(255,  0,255),     // 赤紫
        RGB(255,255,  0),     // 黄色
        RGB(255,255,255)      // 白
};
```

を追加する．

(f) (LatticePCounterDoc.cpp で) OnNewDocument 関数を編集する．

```
BOOL CLatticePCounterDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: この位置に再初期化処理を追加してください。
    // (SDI ドキュメントはこのドキュメントを再利用します。)

    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    // 色を黒に初期化する
    m_nColor = ID_COLOR_BLACK - ID_COLOR_BLACK;

    ////////////////////////////
    // コードの終わり
    ////////////////////////////
```

```
    return TRUE;
}
```

(g) (LatticePCounterDoc.cpp で) AddLinet 関数を編集する.

```
CLine* CLatticePCounterDoc::AddLine(CPoint ptFrom, CPoint ptTo)
{
        // 新しいCLineオブジェクトの生成
        CLine *pLine = new CLine(ptFrom, ptTo, m_crColors[m_nColor]) ;
        try
        {
                // 新しいCLineオブジェクトをオブジェクトの配列に追加
                m_oaLines.Add(pLine) ;
        // ドキュメントに変更が加えられたことを通知するフラグをONにする
                SetModifiedFlag() ;
        }
        // メモリが許容範囲を超えた場合は例外処理を行う
        catch(CMemoryException* perr)
        {
                // 警告メッセージの表示
                AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK) ;
                // CLineオブジェクトを作成したかどうか
                if(pLine)
                {
                        // CLineオブジェクトを解放
                        delete pLine ;
                        pLine = NULL ;
                }
                // 例外オブジェクトを解放
                perr->Delete() ;
        }
        return pLine;
}
```

(h) CLatticePCounterDoc クラスに GetColor 関数を追加する.

(i) [クラスビュー (A)] で [CLatticePCounterDoc] を右クリックし，メニューの中から [追加] − [関数の追加 (U)] を選択する.

(j) 関数の追加ウィザードにおいて，[戻り値の型 (Y)] − [UINT] を選択し，[パラメータの型 (T)] は空白 [] とし,[アクセス (A)] − [public] を選択する.

(k) [関数の名前 (U)] に [GetColor] と入力し，[追加 (A)] ボタンをクリックする．

(l) GetColor 関数を編集する．

```
UINT CLatticePCounterDoc::GetColor(void)
{
        // 選択されている色を返す
        return ID_COLOR_BLACK + m_nColor;
}
```

16. メニューを修正し，メニュー項目 [色 (&C)] を作る．

(a) メニューの [表示 (V)] − [リソースビュー (R)] を選択する．

(b) LatticePCounter リソースのツリーを開き，[Menu] フォルダーのコンテンツを表示させる．[Menu] のコンテンツ [IDR_MAINFRAME] をダブルクリックする．

(c) 一番右の空白のメニューエントリのプロパティを開き [キャプション (C)] に [色 (&C)] と入力して，[メニューアイテムプロパティ] のダイアログを閉じる．

(d) [色 (&C)] メニューエントリの下にサブメニューエントリを追加してプロパティを設定する．

Caption	ID
黒 (&B)	ID_COLOR_BLACK
青 (&L)	ID_COLOR_BLUE
緑 (&G)	ID_COLOR_GREEN
青緑 (&C)	ID_COLOR_CYAN
赤 l(&R)	ID_COLOR_RED
赤紫 (&M)	ID_COLOR_MAGENTA
黄 (&Y)	ID_COLOR_YELLOW
白 (&W)	ID_COLOR_WHITE

(e) メニューバー上でのメニュー [色 (&C)] を [表示 (V)] メニューの前に配置しておく．
イベントハンドラの追加

(f) [色 (&C)] メニューエントリのサブメニューエントリの例えば [黒 (B)] の上で右クリックして [イベントハンドラの追加 (A)] を選ぶ．

(g) イベントハンドラウィザードにおいて，
メッセージの種類 (Y) COMMAND，関数のハンドラ名 (N) OnColorBlack，クラスの一覧で CLatticPCounterDoc を選び，追加して void CLatticePCounterDoc::OnColorBlack() を編集する．
ふたたび，イベントハンドラウィザードにおいて，
メッセージの種類 (Y) UPDATE_COMMAND_UI，関数のハンドラ名 (N) OnUpdateColorBlack，クラスの一覧で CLatticPCounterDoc を選び，追加して void CLatticePCounterDoc::OnUpdateColorBlack(CCmdUI *pCmdUI) を編集する．

(h) 他の色についても同様に行う．

17. メニューを修正し，メニュー項目 [格子間隔 (&P)] を作る．
   (a) メニューの [表示 (V)] − [リソースビュー (R)] を選択する．
   (b) LatticePCounter リソースのツリーを開き，[Menu] フォルダーのコンテンツを表示させる．[Menu] のコンテンツ [IDR_MAINFRAME] をダブルクリックする．
   (c) 一番右の空白のメニューエントリのプロパティを開き [キャプション (C)] に [格子間隔 (&P)] と入力して，[メニューアイテムプロパティ] のダイアログを閉じる．
   (d) [格子間隔 (&P)] メニューエントリの下にサブメニューエントリを追加してプロパティを設定する．
   Caption &5 ID_PITCH_5 CHECKED False
   Caption &10(Default) ID_PITCH_10 CHECKED True
   Caption &20 ID_PITCH_20 CHECKED False
   Caption &40 ID_PITCH_40 CHECKED False
   (e) メニューバー上でのメニュー [格子間隔 (&P)] を [表示 (V)] メニューの前に配置しておく．イベントハンドラの追加
   (f) [格子間隔 (&P)] メニューエントリのサブメニューエントリの例えば [10(Default)] の上で右クリックして [イベントハンドラの追加 (A)] を選ぶ．イベントハンドラウィザードにおいて，
   (g) メッセージの種類 (Y) COMMAND，関数のハンドラ名 (N) OnPitch10，クラスの一覧で CLatticPCounterView を選び，追加して void CLatticePCounterView::OnPitch10() を編集する：

```
void CLatticePCounterView::OnPitch10()
{
        // TODO: ここにコマンド ハンドラ コードを追加します。

        pit = 10;
}
```

ふたたび，イベントハンドラウィザードにおいて，
メッセージの種類 (Y) UPDATE_COMMAND_UI，関数のハンドラ名 (N) OnUpdatePitch10，クラスの一覧で CLatticPCounterView を選び，追加して void CLatticePCounterView::OnUpdatePitch10(CCmdUI *pCmdUI) を編集する：

```
void CLatticePCounterView::OnUpdatePitch10(CCmdUI *pCmdUI)
{
        // TODO: ここにコマンド更新 UI ハンドラ コードを追加します

        ////////////////////////
        // コードの始まり
        ////////////////////////

        // 格子間隔のメニューエントリをチェックするかどうかを決定
        pCmdUI->SetCheck(pit == 10 ? 1 : 0);
```

```
        ////////////////////////
        // コードの終わり
        ////////////////////////
    }
```

(h) 他の格子間隔についても同様に行う.

18. アプリケーション名を適切に修正する.
    MainFrm.cpp の中で CREATESTRUCT cs を修正しておく：

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        if( !CFrameWnd::PreCreateWindow(cs) )
                return FALSE;
        // TODO: この位置で CREATESTRUCT cs を修正して Window クラスまたは
        // スタイルを修正してください。

        cs.style &= ~(FWS_PREFIXTITLE | FWS_ADDTOTITLE);

        return TRUE;
}
```

4 (外部) 関数のコード

ビュークラスのソースファイル LatticePCounterView.cpp は以下のように記述される.

```
// LatticePCounterView.cpp : CLatticePCounterView クラスの実装
//

#include "stdafx.h"
#include "LatticePCounter.h"
#include "MainFrm.h"
#include "Line.h"
#include "LatticePCounterDoc.h"
#include "LatticePCounterView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
#define EPOINTS 25600    //2560
#define INPOINTS 102400    //10240
#define HPOINTS 500    //50


    struct horn{
                    int x0, y0;       // x0 = xl < 0 ,  y0 = yl
                    int x1, y1;       // x1 = xr >= 0 ,  y1 = yr
                    int cx, cy;       // xr*cy - yr*cx = gcd(xr, yr)
                   long px, py;       // px = -cy*xl + cx*yl ,    py = yr*xl - xr*yl > 0
    } HORN = { 0, 0, 0, 0, 0, 0, 0l, 0l}, *pc=&HORN;

    CPoint ptA, m_ptPrevPos;
    CPoint p[EPOINTS]; //CPoint p[25600];
    CPoint q[EPOINTS]; //CPoint q[25600];
    CPoint inp[INPOINTS]; //CPoint inp[102400];

    int nps[HPOINTS]={0}; //int nps[500]={0};
    int vertexN=0, vertexT=0, lineN=0;
    int pit=10, suc, np, hx, pk, fn, flg;    // nsgn 向き付け
    int nline,  intersect=-1,  overlap=-1,  polygoncode=-1;
    long nbp;

   CPoint newp=(100, 100), *w=&newp;
   CPoint vertex=(100, 100), *v=&vertex;
   CPoint coverp=(100, 100), *c=&coverp;
   CPoint frP=(100, 100), *f=&frP;
   CPoint toP=(100, 100), *t=&toP;

    int gcd( int, int);
    int euclid( CPoint *);

   void initpolygon( CPoint [], CPoint [], int [], int);
   void mod_vertex( CPoint, CPoint *, CPoint *);
   void endpoint( int, CPoint []);
    int delvxs( int, int, int, CPoint []);
```

```
    int check_double( int , CPoint []);
   void high_light( int , int *, CPoint []);
   void both_sides( int , struct horn *, CPoint []);
   void check_orientation( int , int *, CPoint []);
   void check_overlap( int , CPoint []);
   void check_intersect( int , CPoint []);
    int line_intersect( CPoint, CPoint, CPoint, CPoint);
   long integral_boundary( int , CPoint [], CPoint []);
    int check_w( int , int , CPoint [], CPoint *);

// CLatticePCounterView

IMPLEMENT_DYNCREATE(CLatticePCounterView , CView)

BEGIN_MESSAGE_MAP(CLatticePCounterView , CView)
// 標準印刷コマンド
ON_COMMAND(ID_FILE_PRINT, &CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, &CView::OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, &CView::OnFilePrintPreview)
ON_WM_LBUTTONDOWN()
ON_WM_RBUTTONDOWN()
ON_WM_RBUTTONUP()
ON_COMMAND(ID_Pitch_5 , &CLatticePCounterView::OnPitch5)
ON_COMMAND(ID_Pitch_10 , &CLatticePCounterView::OnPitch10)
ON_COMMAND(ID_Pitch_20 , &CLatticePCounterView::OnPitch20)
ON_COMMAND(ID_Pitch_40 , &CLatticePCounterView::OnPitch40)
ON_UPDATE_COMMAND_UI(ID_Pitch_5 , &CLatticePCounterView::OnUpdatePitch5)
ON_UPDATE_COMMAND_UI(ID_Pitch_10 , &CLatticePCounterView::OnUpdatePitch10)
ON_UPDATE_COMMAND_UI(ID_Pitch_20 , &CLatticePCounterView::OnUpdatePitch20)
ON_UPDATE_COMMAND_UI(ID_Pitch_40 , &CLatticePCounterView::OnUpdatePitch40)
END_MESSAGE_MAP()

// CLatticePCounterView コンストラクション/デストラクション

CLatticePCounterView::CLatticePCounterView()
: m_ptPrevPos(0)
{
```

```
        // TODO: 構築コードをここに追加します。

}

int gcd( int a, int b)          /*    GCD of a >= b > 0    */
 {
    int s;

    if (a<b) { s = a;           /*    a, b ---> b, a    Interchange    */
                a = b;
                b = s;
    };

    do {
            s = a % b;
            a = b;
            b = s;
    }
    while ( s );

    return a;
 }

int euclid( CPoint *v)
/*    a = v->x, b = v->y : 方程式 a*y - b*x = gcd(a, b) の解を CPoint v に取得する
      a >= b > 0 ---> v->x, v->y satisfying  a*y - b*x = gcd(a, b)    */
{
          int    a = v->x;
          int b = v->y;
          int f=-1, x=0, y=1, x1=-1, y1=0;
        div_t z;

          while (b > 0) {
                z=div(a, b);      /*    z.quot = a / b, z.rem = a % b    */
                a = b;
                b = y;
                y = y1;
```

```
                y1 = -z.quot * y1 + b;
                b = x;
                x = x1;
                x1 = -z.quot * x1 + b;
                b = z.rem;
        };

        v->x = x;
        v->y = y;

        return a;       //   gcd(a, b) ---> a
}


void initpolygon( CPoint p[], CPoint q[], int nps[], int pit)
 {
     int i;

     np=0; suc=0; hx=1; pk=0; fn=-1; flg=-1;       // nsgn=1;
     nline=-1;  intersect=-1;  overlap=-1;  polygoncode=-1;
    (long) nbp=0l;
    (Long) nip=0l;    //  多角形の内部の格子点数の初期値

     // HORN = { 0, 0, 0, 0, 0, 0, 0l, 0l};
     HORN.x0=0; HORN.y0=0; HORN.x1=0; HORN.y1=0;
     HORN.cx=0; HORN.cy=0; HORN.px=0l; HORN.py=0l;

     newp.x=0; newp.y=0;  // newp=(0, 0);

     for (i=0; i<HPOINTS; i++) nps[i]=0;          //#define HPOINTS 500
     for (i=0; i<EPOINTS; i++) p[i] = newp;       //#define EPOINTS 25600
     for (i=0; i<EPOINTS; i++) q[i] = newp;
     for (i=0; i<INPOINTS; i++) inp[i] = newp;    //#define INPOINTS 102400

     return;
 }
```

```
void mod_vertex( CPoint point, CPoint *v, CPoint *c)
/*   Z/pit での頂点の整数座標を v に採り, Z での頂点の整数座標を c に入れる. */
 {
        v->x = point.y / pit;    //  v->x = point.x / pit;
        v->y = point.x / pit;    //  v->y = point.y / pit;
        c->x = pit * (v->y);     //  c->x = pit * (v->x);
        c->y = pit * (v->x);     //  c->y = pit * (v->y);

        return;
 }

int delvxs( int n, int h, int m, CPoint p[])
/*   頂点 p[1], ..., p[n] から, m < h のとき 頂点 p[1], ..., p[m], p[h], ..., p[n]
     を取り除き
     頂点 p[m+1], ..., p[h-1] を残して頂点数を n=h-1-m とする.
     頂点 p[1], ..., p[n] から, h =< m のとき 頂点 p[h], ..., p[m] を取り除き
     頂点 p[1], ..., p[h-1], p[m+1], ..., p[n] を残して 頂点数を n=n-m+h-1 とする.
*/
 {   //   n=np, h, m -> n=np
     //   delete the vertices p[h], ..., p[m].

         int i=1;

         if (m<h) {  //  the vertices p[m+1], ..., p[h-1] .
                    n=h-1-m;
                    while (i<=n) { *(p+i)=*(p+(m+i));
                                   i++;
                    };
         }
         else { i=0;  //  the vertices p[1], ..., p[h-1], p[m+1], ..., p[n] .
                while (i<=n-m-1) {
                                   *(p+(h+i))=*(p+(m+1+i));
                                   i++;
                };
                n=n-m+h-1;
         };
```

```
        return n;
}

void endpoint( int n, CPoint p[])
/*   端末処理  頂点の番号付けを重複化する： p[0] = p[n], p[n+1] = p[1].  */
{    //  n = np,      p[n] = p[0] ,  p[n+1] = p[1]
        *p=*(p+n);
        *(p+(n+1))=*(p+1);
        return;
}

int check_double( int n, CPoint p[])
   /*   二重頂点の除去  Deletion of doubling vertices, n=np ---> n=np   */
{
       int i=1;

       while ( i<=n )
       {  if ( p[i] == p[i+1] )   //   p[i].x==p[i+1].x && p[i].y==p[i+1].y
          {
             n=delvxs(n,i+1, i+1, p);   // 頂点 p[i+1] の除去
             endpoint(n, p);            // 端末処理
          }
          else i++;
       };
       return n;
}

void high_light( int n, int *h, CPoint p[])      //   n = np, h=&hx ---> hx
{    //  最左翼最高頂点 p[h] の取得
        int i=2;

        *h=1;
        while (i<n+1)
        {  if (p[i].x<p[*h].x) { *h=i;}
           else if (p[i].x==p[*h].x && p[i].y>p[*h].y) *h=i;
           i++;
        };
```

```
        return;
 }

void check_intersect( int n, CPoint p[])      //   n = np
    /*   多角形の辺の交差状態 intersect の取得   */
{    //
        int i=2, k, s=-1;

        while (i<n-1){  //  線分 p[n]p[1] と p[i]p[i+1] の交差の確認
                s=line_intersect( p[n], p[1], p[i], p[i+1]);
                if(s==0) intersect=0;
                i++;
         };

        i=1;

        while (i<n-2){
                k=i+2;
                while (k<n){  //  線分 p[k]p[k+1] と p[i]p[i+1] の交差の確認
                        s=line_intersect( p[k], p[k+1], p[i], p[i+1]);
                        if(s==0) intersect=0;
                        k++;
                };

                i++;
        };
        return;
}

int line_intersect( CPoint s, CPoint t, CPoint u, CPoint v)
    /*   二線分の交差状態 w = 0 || -1 の取得   */
{    //
        int z1, z2, w=-1;

  //  点 u, v が直線 st の反対側にあることの確認
        z1 = ((t.x - u.x)*(t.y - s.y) - (t.x - s.x)*(t.y - u.y) )*( (t.x - v.x)*
(t.y - s.y) - (t.x - s.x)*(t.y - v.y) );
```

```
  //  点 s, t が直線 uv の反対側にあることの確認
        z2 = ((v.x - s.x)*(v.y - u.y) - (v.x - u.x)*(v.y - s.y) )*( (v.x - t.x)*
(v.y - u.y) - (v.x - u.x)*(v.y - t.y) );

        if( (z1<0) && (z2<0)) w=0;

        return w;
}

void both_sides( int h, struct horn *pc, CPoint p[])
{   //   最左翼最高頂点 p[h] の両辺 p[h+1]p[h], p[h]p[h-1] の整数成分の取得
        pc->x1=p[h+1].x - p[h].x;           /*   xr >= 0   */
        pc->y1=p[h+1].y - p[h].y;           /*   yr    */
        pc->x0=p[h].x - p[h-1].x;           /*   xl < 0    */
        pc->y0=p[h].y - p[h-1].y;           /*   yl    */
        return;
 }

void check_orientation( int n, int *h, CPoint p[])
/*   多角形の向き付けの確認と Elongated Horn の有無,    nline の取得
          n=np, h=&hx ---> nsgn : Orientation,   多角形の向き付けを正へ
          Elongated-Horn Error ---> nline=0      */
 {
        CPoint sub;
        int i=0;
        long z;

        nline=-1;
        endpoint(n, p);
        high_light(n, h, p);
        both_sides( *h, &HORN, p);

        z=(long) HORN.x1 * HORN.y0 - HORN.x0 * (long) HORN.y1;

        if (z > 0) {      //   向き付け 負    nsgn=-1;
                    for (i=2; i<=(n+1)/2; i++) {
                                                sub=*(p+i);
```

```
                                                *(p+i)=*(p+(n+2-i));
                                                *(p+(n+2-i))=sub;
                        };      //   向き付けを正へ    nsgn=1;
                        endpoint(n, p);
        }
        else {
                //   向き付け 正   nsgn=1;
                if (z==0) nline=0;      //   辺の重畳
        };
        return;
}

void check_overlap( int n, CPoint p[])
/*  頂点の重畳の有無 overlap の取得,  Overlapping-Error ---> overlap = 0 */
{
        int i=1, j, k=-1;

        while ( k && i<=n)
        {       j=i+1;
                while ( k && j<=n)
                {
                   if (p[i]==p[j] ) k=0;   // (p[i].x==p[j].x && p[i].y==p[j].y)
                   j++;
                };
                i++;
        };

        overlap = k;
        return;
}

long integral_boundary( int k, CPoint p[], CPoint q[])
   //   多角形の周上の格子点数 n の取得    k = np ---> n=nbp
   //   多角形の周上の格子点をすべて多角形の頂点化,  CPoint p[] の再定義
{
         int i=1, j, a, a1, b, b1, c;
        long n=0;      //  多角形の頂点数初期値    k = np
```

```
        while (i<k+1)
        {
           a=abs(p[i+1].x-p[i].x);    b=abs(p[i+1].y-p[i].y);
           if ( a>0 && b>0) { c=gcd( a, b);}
           else{ c=max(a, b);};

           a1 = (p[i+1].x-p[i].x)/c;
           b1 = (p[i+1].y-p[i].y)/c;

           for(j=0; j<c; j++){
                                   q[n+1+j].x=p[i].x+a1*j;
                          q[n+1+j].y=p[i].y+b1*j;
           }

           n=n + (long) c ;
           i++;
        };

     for(j=1; j<n+1; j++){
                                   *(p+j)=*(q+j);
     }

     return n;
 }

int check_w( int n, int h, CPoint p[], CPoint *w)
 /*   格子点 w が多角形の頂点 p[1], ... , p[n]  に含まれているかどうかを調べる.
      格子点 w = p[j] のときに, (int) j を返す(1 <= j < h-1 || h+1 < j =< n).
        n=np, h=hx ---> flg=j      */
{
    int i=h+2, j=-1;

    if( h<n-1)
    {
       while (j==-1 && i<=n) {
                          if (p[i].x == w->x && p[i].y == w->y) { j=i;}
```

```
                        else i++;
        };

        i=1;
        while (j==-1 && i<h) { if (p[i].x == w->x && p[i].y == w->y) { j=i;}
                        else i++;
        };
    }
    else{
        if (h==n) { i=2;} else i=1;

        while (j==-1 && i<h) { if (p[i].x == w->x && p[i].y == w->y) { j=i;}
                        else i++;
        };
    }

    return j;
}


CLatticePCounterView::~CLatticePCounterView()
{
}

BOOL CLatticePCounterView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: この位置で CREATESTRUCT cs を修正して Window クラスまたはスタイルを
    // 修正してください。

    return CView::PreCreateWindow(cs);
}

// CLatticePCounterView 描画

void CLatticePCounterView::OnDraw(CDC* pDC)
{
    CLatticePCounterDoc* pDoc = GetDocument();
```

```
ASSERT_VALID(pDoc);
if (!pDoc)
            return;

// TODO: この場所にネイティブ データ用の描画コードを追加します。

////////////////////////////
// コードの始まり
////////////////////////////

// ドキュメント内の直線の数を取得する
int liCount = pDoc -> GetLineCount();

// ドキュメント内に直線が無い場合は処理を行わない
if (liCount)
{
    int liPos;
    CLine *lptLine;

    // 直線の数だけループ
    for (liPos = 0; liPos < liCount; liPos++)
    {
        // ドキュメント内に含まれる CLINE オブジェクトを順番に取得する
        lptLine = pDoc -> GetLine(liPos);

        // 取得した CLINE オブジェクトを描画
        lptLine -> Draw(pDC);
    }
}

//  多角形の内部に格子点がある場合
if( nip>0)
{
        int i;

        //  多角形の内部の格子点の描画
        for (i=1; i<=nip; i++) { pDC -> SetPixel(inp[i], RGB(255, 0, 0));};
```

```
    }


    ////////////////////////
    // コードの終わり
    ////////////////////////
}


// CLatticePCounterView 印刷

BOOL CLatticePCounterView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // 既定の印刷準備
        return DoPreparePrinting(pInfo);
}

void CLatticePCounterView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: 印刷前の特別な初期化処理を追加してください。
}

void CLatticePCounterView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: 印刷後の後処理を追加してください。
}


// CLatticePCounterView 診断

#ifdef _DEBUG
void CLatticePCounterView::AssertValid() const
{
        CView::AssertValid();
}

void CLatticePCounterView::Dump(CDumpContext& dc) const
```

```
{
        CView::Dump(dc);
}

CLatticePCounterDoc* CLatticePCounterView::GetDocument() const
// デバッグ以外のバージョンはインラインです。
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CLatticePCounterDoc)));
        return (CLatticePCounterDoc*)m_pDocument;
}
#endif //_DEBUG


// CLatticePCounterView メッセージ ハンドラ

void CLatticePCounterView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を呼び出します。

    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    //  マウスをキャプチャしているかどうかを確認
        if (GetCapture() == this)
    {
    //  デバイスコンテキスタを取得する
        CClientDC dc(this);

        mod_vertex( point, &vertex, &coverp);
        //  ドキュメントにCLineオブジェクトを追加
        CLine *pLine = GetDocument() -> AddLine(m_ptPrevPos, coverp);

        //  入力された直線を描画
        pLine -> Draw(&dc);

        //  直前の座標として現在の座標を取得
```

```
        m_ptPrevPos = coverp;
        p[vertexN] = vertex;
        vertexN++;
    } else
    {
        //  Document を更新
        CLatticePCounterDoc* pDoc = GetDocument();
        pDoc -> DeleteContents();

        //  View を更新
        Invalidate();

        vertexN=1;
        initpolygon( p, q, nps, pit);

        // マウスの入力を、カーソルの位置にかかわらず全て受け取り、
        // 他のウィンドウに取得されないようにする
        SetCapture();

        mod_vertex( point, &vertex, &coverp);

        ptA = coverp;

        //  直前の座標として現在の座標を取得
        m_ptPrevPos = coverp;    // m_ptPrevPos = point;
        p[vertexN] = vertex;     // p[vertexN] = point;
        vertexN++;
    }

    ////////////////////////
    // コードの終わり
    ////////////////////////

    CView::OnLButtonDown(nFlags, point);
}

void CLatticePCounterView::OnRButtonDown(UINT nFlags, CPoint point)
```

```
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を呼び出します。

    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    //  マウスをキャプチャしているかどうかを確認
    if (GetCapture() == this) {

    //  デバイスコンテキスタを取得する
        CClientDC dc(this);

        mod_vertex( point, &vertex, &coverp);
    //  ドキュメントにCLineオブジェクトを追加
        CLine *pLine = GetDocument() -> AddLine(m_ptPrevPos, coverp);

    //  入力された直線を描画
        pLine -> Draw(&dc);

    //  直前の座標として現在の座標を取得
        m_ptPrevPos = coverp;
        p[vertexN] = vertex;

        vertexT = vertexN;
        lineN = vertexT;

    //  多角形の入力頂点数を確認
        char buffer[20];
        _itoa_s( vertexT, buffer, 10 );
        CString Choten= buffer;
        dc.TextOut(20, 30, "多角図形の入力頂点数  ");
        dc.TextOut(190, 30, Choten);
    };

    ////////////////////////////
    // コードの終わり
```

```
    //////////////////////////

    CView::OnRButtonDown(nFlags, point);
}

void CLatticePCounterView::OnRButtonUp(UINT nFlags, CPoint point)
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を呼び出します。

    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    //  マウスをキャプチャしているかどうかを確認
    if (GetCapture() == this) {

    //  デバイスコンテキスタを取得する
        CClientDC dc(this);

    //  ドキュメントにCLineオブジェクトを追加
        CLine *pLine = GetDocument() -> AddLine(m_ptPrevPos, ptA);

    //  入力された直線を描画
        pLine -> Draw(&dc);     //  入力された直線の数=入力頂点数


        char buffer[20];
        CString Choten= buffer;


    //  多角形の初期頂点数
        np = vertexN;
        endpoint(np, p);
        np = check_double(np, p);       //    二重頂点の除去

        _itoa_s( np, buffer, 10 );
        Choten= buffer;
```

```
    dc.TextOut(20, 50, "多角図形の初期頂点数  "); // 初期頂点数の TextOut
    dc.TextOut(190, 50, Choten);

//  多角形の向き付けの確認と Elongated Horn の有無
    check_orientation(np, &hx, p);
    if(!nline) {  SetTextColor(dc, RGB(255, 0, 0));    // Red
                  dc.TextOut(100, 90, "elongated-horn error!");
    }

//  多角形の周上の格子点数 nbp の取得
    nbp=integral_boundary(np, p, q);
    np = nbp;    //  多角形の周上の格子点はすべて多角形の頂点
    endpoint(np, p);

//  多角形の頂点の重畳の有無の確認
    check_overlap(np, p);
    if(overlap!=-1) { SetTextColor(dc, RGB(255, 0, 0));    // Red
                      dc.TextOut(100, 110, "overlapping error!");
            }

//  多角形の周上の格子点数 nbp の TextOut
    _itoa_s( nbp, buffer, 10 );
    Choten= buffer;
    dc.TextOut(20, 70, "多角図形の辺上の格子点数  ");
    dc.TextOut(220, 70, Choten);

//  多角形の辺の自己交差の確認
    check_intersect( np, p);
    if (intersect!=-1) { SetTextColor(dc, RGB(255, 0, 0));
                         dc.TextOut(100, 130, "Intersect error!");
    };

// マウスをキャプチャしていた場合は、他のウインドウが
// マウスの入力を取得できるようにマウスをリリースする
    ReleaseCapture();
};
```

```
    ////////////////////////
    // コードの終わり
    ////////////////////////

    CView::OnRButtonUp(nFlags, point);
}

void CLatticePCounterView::OnPitch5()
{
        // TODO: ここにコマンド ハンドラ コードを追加します。

        pit = 5;
}

void CLatticePCounterView::OnPitch10()
{
        // TODO: ここにコマンド ハンドラ コードを追加します。

        pit = 10;
}

void CLatticePCounterView::OnPitch20()
{
        // TODO: ここにコマンド ハンドラ コードを追加します。

        pit = 20;

}

void CLatticePCounterView::OnPitch40()
{
        // TODO: ここにコマンド ハンドラ コードを追加します。

        pit = 40;

}
```

```
void CLatticePCounterView::OnUpdatePitch5(CCmdUI *pCmdUI)
{
	// TODO: ここにコマンド更新 UI ハンドラ コードを追加します。

	//////////////////////
	// コードの始まり
	//////////////////////

	// 格子間隔のメニューエントリをチェックするかどうかを決定
	pCmdUI->SetCheck(pit == 5 ? 1 : 0);

	//////////////////////
	// コードの終わり
	//////////////////////
}

void CLatticePCounterView::OnUpdatePitch10(CCmdUI *pCmdUI)
{
	// TODO: ここにコマンド更新 UI ハンドラ コードを追加します。

	//////////////////////
	// コードの始まり
	//////////////////////

	// 格子間隔のメニューエントリをチェックするかどうかを決定
	pCmdUI->SetCheck(pit == 10 ? 1 : 0);

	//////////////////////
	// コードの終わり
	//////////////////////
}

void CLatticePCounterView::OnUpdatePitch20(CCmdUI *pCmdUI)
{
	// TODO: ここにコマンド更新 UI ハンドラ コードを追加します。

	//////////////////////
```

```
        // コードの始まり
        //////////////////////

        // 格子間隔のメニューエントリをチェックするかどうかを決定
        pCmdUI->SetCheck(pit == 20 ? 1 : 0);

        //////////////////////
        // コードの終わり
        //////////////////////
}

void CLatticePCounterView::OnUpdatePitch40(CCmdUI *pCmdUI)
{
        // TODO: ここにコマンド更新 UI ハンドラ コードを追加します。

        //////////////////////
        // コードの始まり
        //////////////////////

        // 格子間隔のメニューエントリをチェックするかどうかを決定
        pCmdUI->SetCheck(pit == 40 ? 1 : 0);

        //////////////////////
        // コードの終わり
        //////////////////////
}
```

LatticePCounterView.cpp における上のような記述を通して，**仕様　格子多角形の取得** ができる アプリケーションが得られる．

5　格子多角形の内点の取得に関する仕様の実装

Windows アプリケーション LatticePCounter に格子多角形の内点すべての位置取得の機能を与える．

仕様　格子多角形の内点の取得

- 多角形 P：$P_1P_2, P_2P_3, \cdots, P_nP_1$ 内の格子点をの位置を取得し，数え上げる．
- 多角形 P 内の格子点をプロットする．

詳しい手順は以下の通りである．

LatticePCounterView.cpp に多角形 P の内部の格子点数を入れる変数と関数を追加する．

```
long nip;    //  多角形の内部の格子点数
 int left_edge(int, int *, struct horn *, CPoint []);
void horn_reduction( struct horn *);
void getatarget( struct horn *);
 int loadbackward( int, int, int, CPoint []);
void searchvx( int, struct horn *, CPoint [], CPoint *);
 int putvx( int, int, int [], struct horn *, CPoint [],
            CPoint [], CPoint *);
 int contraction11( int, int, CPoint []);
 int contraction12( int, int, CPoint []);
 int contraction21( int , int , int , CPoint []);
 int contraction22( int, int, int, int [], CPoint [], CPoint []);
 int subpolygon( int, int, int, int [], CPoint [], CPoint []);
```

LatticePCounterView.cpp の中で，関数

```
int check_w( int n, int h, CPoint p[], CPoint *w)
{
   ...
}
```

の後に，に関数のコードを追加する．

```
int left_edge( int n, int *h, struct horn *pc, CPoint p[])
     /*    最左翼最低辺上の格子点の挿入
           n = np, h=&hx ---> n=np-1+z, hx!=1--->hx=hx-1+z
           Commensurable Edge (pc->x0, pc->y0) = (xl, yl)/z    */
```

```
{
    int i, z=1;

    if ( pc->x0!=-1 && pc->y0!=1 && pc->y0!=-1)
                        {
                          if (pc->x0!=0 && pc->y0!=0) {
                                                        z=gcd(-pc->x0, abs(pc->y0));
                                                      }
                          else {
                                  z=max(-pc->x0, abs(pc->y0));
                          }
    };
                                /*   z=gcd of the edge   */
    pc->x0=pc->x0 / z;          /*   x-component of Commensurable Edge  */
    pc->y0=pc->y0 / z;          /*   y-component of Commensurable Edge  */

    if (z>1)
    {
              if (*h>1)
              {
                        n=loadbackward(n, *h-1, z-1, p);  /* 多角形の周上の
                        頂点 p[h+1], ..., p[n] の番号付けを z-1 番後送  */

                        for (i=*h-1; i<*h+z-1; i++) {
                                                      p[i+1].x=p[i].x + pc->x0;
                                                      p[i+1].y=p[i].y + pc->y0;
                        };     /*  多角形の周上の頂点
                                          p[h+1], ..., p[h+z-1] の挿入  */
                        *h=*h+z-1;
              }
              else {    //  多角形の周上の頂点 p[n+1], ..., p[n+z-1] の挿入
                        for (i=n; i<n+z-1; i++) {
                                                    p[i+1].x=p[i].x + pc->x0;
                                                    p[i+1].y=p[i].y + pc->y0;
                        };

                        n=n+z-1;
```

```
            };

                endpoint(n, p);
    };

      return n;
}

void horn_reduction( struct horn *pc)
/*  構造体 horn の簡約可能性 : pc->x1, pc->y1 の最大公約数 a の取得
    Data pc->cx, pc->cy : 方程式 (pc->x1)*cy - (pc->y1)*cx = a の解を取得する.
    構造体 horn の簡約 ---> gcd(pc->x1, pc->y1) = 1   */
{
    int a, f=-1;

    if (pc->x1>=abs(pc->y1)) {
                                v->x=pc->x1;
                                v->y=abs(pc->y1);
    }
    else {
        v->x=abs(pc->y1);
        v->y=pc->x1;
        f=0;
    };

    a = euclid( v);       //   pc->x1, abs(pc->y1) の最大公約数 a の取得

    pc->x1=pc->x1/a;      //   構造体 horn pc の簡約化
    pc->y1=pc->y1/a;

    if ( f ) {
            pc->cx = v->x;
            pc->cy = v->y;
    }
    else {
        pc->cx = - v->y;
        pc->cy = - v->x;
```

```
        };

        return;
}

void getatarget( struct horn *pc)
/*    構造体 horn *pc=&HORNの簡約可能性
          pc->x1, pc->y1 の最大公約数 gcd = a の取得
          Data pc->cx, pc->cy :
          方程式 (pc->x1)*cy -  (pc->y1)*cx = a の解を取得 ---> pc->cx, pc->cy
          構造体 horn の簡約 : 通約辺 (pc->x1, pc->y1) = (xr, yr)/a      */
    /*    horn 内の target の取得 pc->cx, pc->cy ---> pc->px, pc->py    */
    /*    winding error ---> pc->py=0    */
  {
          if (pc->x1==1 || pc->y1==0) {
                                        if (pc->y1==0) pc->x1=1;
                                        pc->cx=0;
                                        pc->cy=1;
          }
          else {
                  if (pc->x1==0 || pc->y1==-1) {
                                                 if (pc->x1==0) pc->y1=-1;
                                                 pc->cx=1;
                                                 pc->cy=0;
                  }
                  else {
                          if (pc->y1==1) { pc->cx=-1;
                                           pc->cy=0;
                          }
                          else {
                                  if (pc->x1==abs(pc->y1)) { pc->x1=1;
                                                             if (pc->y1<0) { pc->y1=-1;}
                                                             else pc->y1=1;
                                                             pc->cx=0;
                                                             pc->cy=1;
                                  }
                                  else {
```

```
                                        horn_reduction(pc);
                                        if (pc->y1<0) pc->cx=-pc->cx;
                                }
                        }
                }
        };

        pc->px = (long) -(pc->cy) * (pc->x0) + (long) (pc->cx) * (pc->y0);
        pc->py = (long) (pc->y1) * (pc->x0) - (long) (pc->x1) * (pc->y0);
        //    pc->py > 0

        return;
}

int loadbackward( int n, int h, int m, CPoint p[])
/*    多角形の周上の頂点 p[h+1], ..., p[n] の番号付けを m 番後送
      p[h+1+m], ..., p[n+m] とする.       n=np ---> n=np+m, h+1 ---> h+1+m
*/
 {
        int i=n;

        while (i>h) {
                        *(p+(i+m))=*(p+i);
                        i--;
        };
        n=n+m;
        return n;
}

void searchvx( int h, struct horn *pc, CPoint p[], CPoint *w)
      /*    horn 内の target から多角形に含まれている格子点 w を取得する.
            h=hx, pc=&HORN, pc->py>0 ---> the vertex newp    */
{
  if (pc->py==1) { pk = pc->px;
                        *w=p[h-1];
  }
  else {  if (pc->px<=0l) { pk = (pc->px)/(pc->py);}
```

```
            else pk = (pc->px)/(pc->py)+1;
            w->x=p[h].x + pc->cx + pk*(pc->x1);
            // w->x - p[h].x = pc->cx + pk*(pc->x1) > 0
            w->y=p[h].y + pc->cy + pk*(pc->y1);
    };
    return;
}

int putvx( int n, int h, int nps[], struct horn *pc, CPoint p[],
            CPoint q[], CPoint *w)
/*    多角形の周上の格子点がすべて多角形の頂点であるとき!
                n=np, h=hx, pc->py > 0
            多角形に含まれている格子点 newp から, 新たな多角形を構成し還元処理
*/
 {
      fn=-1;
      int pin =1;

      if (pc->py==1) {    // このときに限り,  newp = p[h-1]
                        if ( n<=3) { n=2;}
                        else { if ( pc->x1==0) { n=contraction11(n, h, p);}
                                    else n=contraction12(n, h, p);
                        }
      }
      else { flg=check_w(n, h, p, w);

              if (flg != -1)
              {
                      if( (flg==(h+2) ) || (h==n && flg==2) || (h==n-1 && flg==1) )
                      {       n=contraction21(n, h, flg , p);}
                      else{  n=contraction22(n, h, flg , nps, p, q);}
              }
              else {    // flg = -1
                       nip++;      //  多角形の内部の格子点数
                       inp[nip].x = pit*(w->y);
                       inp[nip].y = pit*(w->x);
```

```
                    n=loadbackward( n, h, pin, p);
                    *(p+(h+1)) = *w;

                    endpoint( n, p);
                    fn=3;
            };
    };

    return n;
}

int contraction11( int n, int h, CPoint p[])
    /*   pc->py==1 (このとき, newp = p[h-1]), pc->x1==0 の場合  */
    /*   n=np>3, h=hx ---> n=np, hx, fn  */
{
    int k=h;

    while (p[k+1].x==p[k].x) {
                                if (k==n) { k=1;} else k++;
    };

    n=delvxs(n, h, k-1, p);

    fn=-1;

    return n;
}

int contraction12( int n, int h, CPoint p[])
    /*   pc->py==1 (このとき, newp = p[h-1]), pc->x1>0 の場合  */
    /*   n=np>3, h=hx ---> n=np, hx, fn  */
{
       n=delvxs(n, h, h, p);
       endpoint(n, p);

       fn = -1;
```

```
    return n;
}

int contraction21( int n, int h, int m, CPoint p[])
/*   n=np, h=hx, m=flg (このとき, newp = p[m]) ---> n=np, hx,  fn=1
                          gcd(p[hx+1].x-p[hx].x, abs(p[hx+1].y-p[hx].y))=1
*/
{
    if ( m==h+2 || (m==2 && h==n)) { n=delvxs(n, m-1, m-1, p);
                                     if (m==2 && h==n+1) hx=n;
    }
    else {
            if (m==1 && h==n-1) { n--;
                                  hx=n;
            }
    };

    endpoint(n, p);
    fn=3;

    return n;
}

int contraction22( int n, int h, int m, int nps[], CPoint p[], CPoint q[])
/*   n=np, h=hx, m=flg (このとき, newp = p[m]) ---> n=np, hx, fn=1
                          gcd(p[hx+1].x-p[hx].x, abs(p[hx+1].y-p[hx].y))=1
*/
{
    n=subpolygon(n, h, m, nps, p, q);   //  二つの多角形へ分解  suc++
    endpoint(n, p);

    if (h>m) hx=n;
    fn=3;

    return n;
}
```

```
int subpolygon( int n, int n0, int n1, int nps[], CPoint p[], CPoint q[])
/*   n=np, n0=hx, newp=p[n1=flg] ---> n=np, p[], q[]
                部分多角形を取得し，処理する. Flag suc を +1.   */
/*   n0 < n1 のとき，頂点 p[n0+1], ..., p[n1] を q[nps[suc]+1], ...,
     q[nps[suc]+n1-n0] に取得し  nps[suc+1]=nps[suc]+n1-n0 とする.
     多角形 P から頂点 p[n0+1], ..., p[n1-1] を取り除く.
     n0 >= n1 のとき，頂点 p[1], ..., p[n1], p[n0+1], ..., p[n] を
     q[nps[suc]+1], ..., q[nps[suc]+n-n0+n1]  に取得
     nps[suc+1]=nps[suc]+n-n0+n1 とする.
     多角形 P から頂点 p[1], ..., p[n1-1], p[n0+1], ..., p[n] を取り除く.
*/
{
     int i=0;

     if (n0<n1) {
                for (i=1; i<=n1-n0; i++) {
                                        *(q+(nps[suc]+i))=*(p+(n0+i));
                };
                nps[suc+1]=nps[suc]+n1-n0;
     }
     else {
            for (i=1; i<=n1; i++) {
                                *(q+(nps[suc]+i))=*(p+i);
            };
            for (i=n1+1; i<=n-n0+n1; i++) {
                                        *(q+(nps[suc]+i))=*(p+(n0+i-n1));
            };
            nps[suc+1]=nps[suc]+n-n0+n1;
           };

     suc++;
     n=delvxs(n, n0+1, n1-1, p);          /*   if (n1<n0) flg=n;   */
     return n;
}
```

```
void CLatticePCounterView::OnRButtonUp(UINT nFlags, CPoint point)
{
    // TODO: ここにメッセージ ハンドラ コードを追加するか、既定の処理を呼び出します。

    ////////////////////////////
    // コードの始まり
    ////////////////////////////

    ...

    //  多角形の辺の自己交差の確認
        check_intersect( np, p);
        if (intersect!=-1) { SetTextColor(dc, RGB(255, 0, 0));
                             dc.TextOut(100, 130, "Intersect error!");
        };

    //  多角形の内部の格子点数 nip の取得
        int i;

        do {
             if (suc>0) { np=nps[suc]-nps[suc-1];
                          for (i=1; i<=np; i++) { *(p+i)=*(q+(nps[suc-1])+i);};
                          suc--;
                          };

                          while (np>2)
                          {       /*   fn = 3, -1   */
                             if (fn==-1) {
                                          high_light(np, &hx, p);
                                          endpoint(np, p);
                             };
                             both_sides(hx, &HORN, p);
                             np=left_edge(np, &hx, &HORN, p);
                             getatarget( &HORN);

                             if (pc->py<=0l) polygoncode=0;
```

```
                        if (polygoncode!=-1)
                        {  //  多角形の頂点の回転?の確認
                           SetTextColor(dc, RGB(255, 0, 0));  // Red
                           dc.TextOut(100, 150, "winding error!");
                           break;
                        };

                        searchvx(hx, &HORN, p, &newp);

                        //  多角形の内部の格子点数 nip の取得
                        np=putvx(np, hx, nps, &HORN, p, q, &newp);
        };
    } while (suc>0);

//  多角形の内部の格子点の描画
    for (i=1; i<=nip; i++) { dc.SetPixel(inp[i], RGB(255, 0, 0));};

    //  多角形の内部の格子点数 nip の TextOut
    if ((polygoncode)&&(intersect)&&(overlap)) {
                        _itoa_s( nip, buffer, 10 );
                        Choten = buffer;
                        dc.TextOut(20, 180, "多角形の内部の格子点数 = ");
                        dc.TextOut(220, 180, Choten);

                        _itoa_s(nbp, buffer, 10);
                        Choten = buffer;
                        dc.TextOut(20, 160, "多角形の辺上の格子点数 = ");
                        dc.TextOut(220, 160, Choten);
         }
    else {
          if (polygoncode) {
                _itoa_s( 0l, buffer, 10 );
                Choten= buffer;
                dc.TextOut(20, 180, "多角図形の内部の格子点数  ");
                        dc.TextOut(220, 180, Choten);
    };
```

```
    // マウスをキャプチャしていた場合は、他のウインドウが
    // マウスの入力を取得できるようにマウスをリリースする
        ReleaseCapture();
    };

    ////////////////////////////
    // コードの終わり
    ////////////////////////////

    CView::OnRButtonUp(nFlags, point);
}
////////////////////////////////////////////////
```

付録　定理１．の証明

定理１．多角形 P の Horn $\angle SAB$ を考える; S, A, B は多角形 P の頂点である．
多角形 P の頂点 S, A, B の順序は多角形 P の正の向きに合っている．この時、以下のことが成り立つ．
(1)　辺 AB 上の格子点 $E\ (\neq A)$ を線分 AE 上には A, E 以外の格子点が存在しない様に取り，

$\overrightarrow{AE} = (x_1, y_1) \in \mathbf{Z}^2$ とする．このとき，$x_1 \geq 0$ で最大公約数 GCD $(x_1, y_1) = 1$ であり，方程式

$$x_1 \cdot y - y_1 \cdot x = 1 \quad (x, y \in \mathbf{Z})$$

の１つの整数解 $(cx, cy) \in \mathbf{Z}$ が存在する；$\overrightarrow{AD} = (cx, cy) \in \mathbf{Z}^2$ とする．
格子点 D は格子点 A, E, B を通る直線 $l_0 : t\overrightarrow{AE}\ \ (\forall t \in \mathbf{R})$ に平行な直線 l_1

$$l_1 \quad : \quad t\overrightarrow{AE} + \overrightarrow{AD} \qquad (\forall t \in \mathbf{R})$$

上にある（格子点 A を原点 $\boldsymbol{O}$ として考えても，この定理の主張に問題は生じない）．

(2)　辺 SA 上の格子点 $R\ (\neq A)$ を，線分 RA 上には A, R 以外の格子点が存在しない様に取り，

$$\overrightarrow{RA} = (x_0, y_0) \in \mathbf{Z}^2$$

とおくと，　$x_0 < 0$.
このとき　$x_1 \cdot cy - y_1 \cdot cx = 1$ であるから，連立方程式

$$\begin{cases} -x_0 = x_1 \cdot x + cx \cdot y \\ -y_0 = y_1 \cdot x + cy \cdot y \end{cases}$$

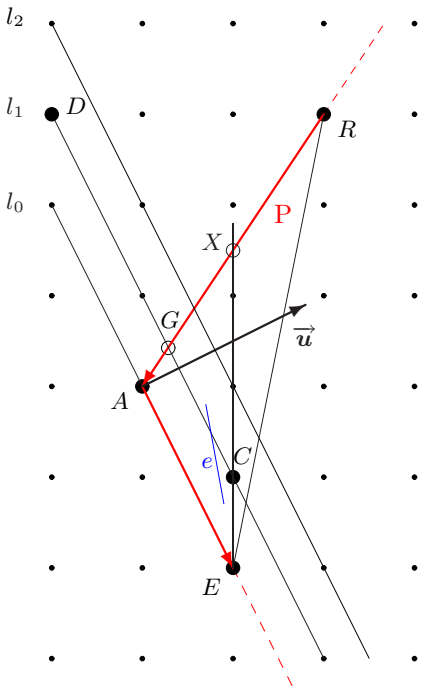
は整数解 $x = px,\ y = py$ を持ち，

$$\begin{cases} px = -x_0 \cdot cy + y_0 \cdot cx \\ py = -x_1 \cdot y_0 + y_1 \cdot x_0 \ > 0 \end{cases}$$

が成り立つ．

$$m = \begin{cases} px & (py = 1 \text{ の場合}) \\ \left[\dfrac{px}{py}\right] + 1 & (py > 1 \text{ の場合}) \end{cases}$$

と置く．
（ここで，Gauss 記号　$[a] = \max\{n \in \mathbf{Z} \ : \ n \leq a\}$.）



この時，$\overrightarrow{AC} = m\overrightarrow{AE} + \overrightarrow{AD}$ となる格子点 C に対して，$\triangle AEC$ は多角形 P に含まれる面積 $\dfrac{1}{2}$ の三角形で，その内部と辺上には（頂点以外）格子点がない．

(*) 注意 1．定理１．において，格子点 C は多角形 P の辺周上の格子点であることもある．
(*) 注意 2．定理１．において，$x_1 = 0$ のとき $y_1 = -1$ である．この場合には，$cx = 1,\ cy = 0$ とすることができて，$px = y_0,\ py = -x_0 > 0$ が成り立つ．

証明.

[I] 最初に，次のことに注意する．任意の整数 α, β に対する連立方程式

$$\begin{cases} \alpha = x_1 \cdot x + cx \cdot y \\ \beta = y_1 \cdot x + cy \cdot y \end{cases}$$

は解

$$\begin{cases} x = \alpha \cdot cy - \beta \cdot cx \\ y = x_1 \cdot \beta - y_1 \cdot \alpha \end{cases}$$

を持つから，$\mathbf{Z}^2 = \left\{ a\overrightarrow{AE} + b\overrightarrow{AD} \;\middle|\; \forall a, b \in \mathbf{Z} \right\}$ である．

$$\begin{aligned} \overrightarrow{EC} &= \overrightarrow{RC} - \overrightarrow{RE} \\ &= \left(\overrightarrow{RA} + \overrightarrow{AC}\right) - \left(\overrightarrow{RA} + \overrightarrow{AE}\right) \\ &= \left(m\overrightarrow{AE} + \overrightarrow{AD}\right) - \overrightarrow{AE} \\ &= (m-1)\overrightarrow{AE} + \overrightarrow{AD} \\ &= \left[\frac{px}{py}\right]\overrightarrow{AE} + \overrightarrow{AD} \end{aligned}$$

であるから，

$$\begin{aligned} \mathbf{Z}^2 &= \left\{ a\overrightarrow{AE} + b\overrightarrow{AD} \;\middle|\; \forall a, b \in \mathbf{Z} \right\} \\ &= \left\{ a\overrightarrow{AE} + b\left(\overrightarrow{EC} - \left[\frac{px}{py}\right]\overrightarrow{AE}\right) \;\middle|\; \forall a, b \in \mathbf{Z} \right\} \\ &= \left\{ a'\overrightarrow{AE} + b'\overrightarrow{EC} \;\middle|\; \forall a', b' \in \mathbf{Z} \right\} \end{aligned}$$

であることに注意する．すなわち，ベクトルの組 $\left\{\overrightarrow{AE},\ \overrightarrow{AD}\right\}$ も $\left\{\overrightarrow{AE},\ \overrightarrow{EC}\right\}$ も，それぞれ加群 $\mathbf{Z}^2$ の生成元である．

[II] 三角形 $\triangle AEC$ の面積は，（A を原点として考えて）つぎのように計算できる：

$$\frac{1}{2}\begin{vmatrix} x_1 & mx_1 + cx \\ y_1 & my_1 + cy \end{vmatrix} \;=\; \frac{x_1 \cdot cy - y_1 \cdot cx}{2} \;=\; \frac{1}{2}.$$

Pick の定理から，格子点を頂点とする三角形 $\triangle AEC (= \triangle AER)$ の内部と辺上には頂点以外の格子点は存在しないことがわかる．

さて，格子点 A で $\overrightarrow{AE}$ と直交するベクトル $\overrightarrow{\boldsymbol{u}} = (-y_1, x_1)$ を考えると，補題から，内積

$$\overrightarrow{AR} \cdot \overrightarrow{\boldsymbol{u}} = x_0 \cdot y_1 - x_1 \cdot y_0 \;= py > 0.$$

今，直線 l_1 と直線 AR の交点を G とすると，

$$l_1 \quad : \quad t\overrightarrow{AE} + \overrightarrow{AD} \qquad (\forall t \in \mathbf{R})$$

で　$\overrightarrow{AR}\ (= -\overrightarrow{RA}) = px \cdot \overrightarrow{AE} + py \cdot \overrightarrow{AD}$　となっているから，

$$\overrightarrow{AG} = \frac{px}{py} \cdot \overrightarrow{AE} + \overrightarrow{AD}.$$

これから，点 G は線分 RA 上にあることがわかる．

$py = 1$ の場合には，線分 AR 上には A, R 以外の格子点が存在しないので，G は格子点で $G = C = R$ である．この場合，三角形 $\triangle AEC (= \triangle AER)$ は多角形 P に含まれることはつぎのように考えてわかる．三角形 $\triangle AEC = AER$ が多角形 P に含まれないと仮定しよう．そのとき，三角形 $\triangle AER$ の内部を通る（かつ三角形 $\triangle AER$ の頂点を通らない）多角形 P の辺 e がなければならないが，辺 e の端点（格子点）は三角形 $\triangle AER$ の内には存在しえないので三角形 $\triangle AER$ のある一辺と交わる．この場合，Pasch の公理より，三角形 $\triangle AER$ の他の辺 RA, AE のどちらかとも交わることになる－これは P が多角形であることに矛盾する．

$py > 1$ の場合，線分 RA 上には A, R 以外の格子点が存在しないから，最大公約数 GCD $(px, py) = 1$ で

$$1 > \left[\frac{px}{py}\right] + 1 - \frac{px}{py} > 0$$

が成り立っている．従って

$$\overrightarrow{GC}\ (=\ \overrightarrow{AC} - \overrightarrow{AG})\ =\ \left(\left[\frac{px}{py}\right] + 1 - \frac{px}{py}\right) \cdot \overrightarrow{AE}$$

から，C は直線 RA に関して格子点 E とは同じ側に在ることがわかる．頂点 E を端点とする半直線 EC 上の点 $\overrightarrow{AE} + k\overrightarrow{EC}$ $(k \in \mathbf{N})$ はすべて格子点であり，（方程式 $x_1 \cdot y - y_1 \cdot x - k = 0$ で決まる）直線

$$l_k \quad : \quad t\overrightarrow{AE} + k\overrightarrow{AD} \qquad (\forall t \in \mathbf{R})$$

上にある．

$$\begin{aligned} \because \qquad \overrightarrow{AE} + k\overrightarrow{EC} &= \overrightarrow{AE} + k\left(\left[\frac{px}{py}\right]\overrightarrow{AE} + \overrightarrow{AD}\right) \\ &= \left(1 + k\left[\frac{px}{py}\right]\right)\overrightarrow{AE} + k\overrightarrow{AD}, \qquad \text{ここで} \quad 1 + k\left[\frac{px}{py}\right] \ \in\ \mathbf{Z}. \end{aligned}$$

ベクトルの組 $\left\{\overrightarrow{AE},\ \overrightarrow{EC}\right\}$ は加群 $\mathbf{Z}^2$ の生成元であるから，線分 AE と EC を辺とする平行四辺形の内部（この平行四辺形からその辺周を除いた部分）には格子点がなく，さらにこの平行四辺形の $\mathbf{Z}^2$ の元による平行移動の内部にも格子点はないいことがわかる．今，半直線 EC と直線 RA との交点を X とすると，三角形 $\triangle XAE$ に含まれる格子点は頂点 A, E と線分 EX 上の格子点だけである．辺 RA 上には A, R 以外の格子点はないから，線分 XA $(\subset RA)$ となっていることがわかる．

もし格子点 C が多角形 P の外点ならば，三角形 $\triangle AEC$ は多角形 P に含まれないので，三角形 $\triangle AEC$ の内部を通り格子点 A, C を通らない（かつ線分 AE と異なる）多角形 P の辺 e がなければならないが，この辺 e の端点（格子点）の一つは頂点 E であるか，またはこの辺 e の両端点は三角形 $\triangle AEC$ を含む三角形 $\triangle XAE$ の外部にある．

e の端点の一つが頂点 E である場合には， P の辺 e が辺 XA $(\subset RA)$ と交わり， P が多角形であることに矛盾する．

辺 e の両端点が三角形 $\triangle AEC$ を含む三角形 $\triangle XAE$ の外部にある場合には，辺 e は三角形 $\triangle XAE$ のある一辺と交わる．この場合，Pasch の公理より，三角形 $\triangle XAE$ の他の辺 XA $(\subset RA)$, AE のどちらかとも交わることになる－これは P が多角形であることに矛盾する．従って，格子点 C は多角形 P に含まれる．

Q.E.D.

「LatticePCounter」プログラムは，Pick の定理を Test するためのプログラムである．もちろん，格子点を頂点とする任意の平面多角形の辺周上の頂点の情報からその辺周上および内部の格子点数を知る手続きの適切さを Test している．特に，格子点を頂点とする任意の閉経路 P：$P_1, P_2, \cdots, P_n, P_1$ が自己交差のない真の多角形 P のとなっているかどうかを判定するプログラムであるかを Test しているともいい得る．

Windows アプリケーションとしてのプログラム「LatticePCounter」は，[3] を参考/引用して，CLine クラスの作成とクラスへの関数の追加，Cview クラスへのコードの追加で作成することができた．思えば，所謂 Polygon Class に多角図形の Horn を要素として設定することも考えられるだろうか？

思い出として 2013/07/07（& 加筆 2019.02.16)

参考文献

[1] デービス・チャップマン，青山ひろあき監訳：3 週間完全マスター VisualC++ 6.0，日経 BP ソフトプレス，1999 年.

[2] 林晴比古：明快入門 Visual C++ 2005 ビギナー編，ソフトバンククリエイティブ株式会社，2006 年.

[3] 浦田敏夫：Pick の定理をめぐって; 愛知教育大学数学教育学会誌イプシロン, 34 (1992), 85-98.

[4] Hammer J. : Unsolved problems concerning lattice points, Research Notes in Math. 15, Pitman, London·San Francisco·Melbourne, 1977.

[5] F. P. Preparata & M. l. Shamos : Computational Geometry An Introduction, Springer-Verlag GmbH & Co. KG.,1988. (浅野孝夫/浅野哲夫訳 ： 計算幾何学入門, 総研出版, 1992)

-VisualC++ 2008 **による**- LatticePointCounter **の作成**