РЕКОМЕНДАЦИИ ДЛЯ ВЫПОЛНЕНИЯ ЗАДАНИЯ 8

Функциональность, заложенная при выполнении задания 7 должна сохраниться в настоящем проекте. Поэтому, для начала создадим копию проекта созданного в седьмом задании и будем поэтапно вносить изменения в этот проект.

Изменения в формате входного файла

Пусть входной файл может содержать все команды, декларированные в задании 7. Кроме этого добавим возможность указания команды для указания цвета

```
color R G B
```

в которой `R`, `G`, `B` — целочисленные значения от 0 до 255 — количество красной, зеленой и синей составляющей заданного цвета, соответственно. Указание этой команды во входном файле означает, что последующие многоугольники должны иметь указанный цвет (пусть цвет по умолчанию — черный).

Организуйте сохранение цвета для каждого многоугольника.

Измените процедуру отрисовки каждого многоугольника (в процедуре *Form1_Paint*) таким образом, чтобы вычерчивание линий проводилось пером с цветом многоугольника. Для этого будет полезной функция `Color::FromArgb(R, G, B)`, возвращающая значение типа `Color` для цвета составленного из красной, зеленой и синей составляющей.

Извлечение цвета пиксела на форме

В обработчике события *Form1_Paint* проведем следующие изменения. Сопоставим значение переменной `g` не с формой, а с растровым изображением, имеющим размеры формы, т.е. вместо

```
Graphics^ g = e->Graphics;
```

напишем

```
Bitmap^ image1  = gcnew  Bitmap( this->ClientRectangle.Width
                                , this->ClientRectangle.Height
                                );
Graphics^ g = Graphics::FromImage(image1);
```

Теперь все наши действия, связанные с рисованием на графической плоскости `g`, будут выполняться не на форме, а на изображении `image1` (находящемся в памяти). Для того, чтобы изображение появилось на форме, нужно в конце процедуры (перед закрытием процедуры) добавить

```
g = e->Graphics;
g->DrawImage(image1,0,0);
delete image1;
```

т. е. сначала переприсваивается значение `g`: переменная связывается с областью рисования на форме, а после этого на форму выводится изображение сформированное в `image1`, после чего удаляется объект, связанный с `image1`, созданный в начале процедуры.

Теперь, перед тем, как вывести изображение `image1` на форме, можно извлекать и устанавливать новые значения цвета пикселов в изображении `image1`.

Например, чтобы узнать значение цвета пиксела с координатами `(x, y)`, его нужно извлечь с помощью процедуры `GetPixel(x, y)` и сохранить в переменной типа `Color`.

```
Color pixelColor = image1->GetPixel(x, y);
```

Для того, чтобы сравнить значение полученного цвета с определенным цветом, лучше переводить значения цвета в тип `int` с помощью функции `ToArgb`, например, чтобы сравнить, что значение цвета — черный цвет:

```
 if (pixelColor.ToArgb() == Color::Black.ToArgb()) {  }
```

Для того, чтобы изменить значение цвета пиксела на другой цвет, нужно использовать процедуру `SetPixel`, например чтобы изменить значение цвета на красный цвет:

```
image1->SetPixel(x, y, Color::Red);
```

Нахождение внутренней точки многоугольника

В алгоритмах заливки области требуется иметь начальную точку, принадлежащую внутренности этой области. В задании 8 областью закраски является многоугольник. Рассмотрим, как можно найти внутреннюю точку многоугольника.

Пусть задан многоугольник списком своих вершин.

Сначала найдем прямую, пересекающую многоугольник, но на которой не лежит ни одна из вершин многоугольника. Для этого найдем два различных минимальных значения координаты y вершин многоугольника. Обозначим их y' и y''. Пусть $y' < y''$. Возьмем прямую

$$y = \frac{y'' - y'}{2}.$$

Найдем точки пересечения этой прямой с ребрами многоугольника. Очевидно, что эта прямая будет пересекаться с ребрами, имеющими координату y одной из вершин равную y' (горизонтальные ребра стоит исключить из рассмотрения). Таких точек пересечения не менее 2-х. Найдем два минимальных значения координаты x для таких точек пересечения. Обозначим их x' и x''. Пусть $x' < x''$. Точка $\left(\frac{x'' + x'}{2}, \frac{y'' + y'}{2}\right)$ является внутренней точкой многоугольника[1].

Реализация алгоритма закраски многоугольника

В файле `Pclip.cpp` реализуйте процедуру

```
void Pfill (polygon^ P, System::Drawing::Bitmap^ image, System::Drawing::Color C)
```

организующую закраску цветом `C` многоугольника `P`, границы которого изображены на `image` цветом `C`. Здесь предполагается, что многоугольник `P` — результат работы алгоритма отсечения многоугольника.

Процедура должна вычленить из списка `P` каждую видимую нетривиальную часть многоугольника (видимую часть многоугольника `P` без самопересечений и вырожденных ребер) и запустить для нее алгоритм закраски, соответствующий вашему варианту.

В случае алгоритмов заливки области процедура должна предварительно перед закраской организовать поиск внутренней точки каждой видимой части.

[1] Данный метод может не сработать если существует часть многоугольника вырожденная в отрезок, начинающийся в точке с координатой y'. Такой случай можно обойти с помощью введения дополнительных проверок.