РЕКОМЕНДАЦИИ ДЛЯ ВЫПОЛНЕНИЯ ЗАДАНИЯ 6

Функциональность, заложенная при выполнении задания 4 должна сохраниться в настоящем проекте. Поэтому, для начала создадим копию проекта созданного в четвертом задании и будем поэтапно вносить изменения в этот проект.

1 Отображение прямоугольника и отсечение изображения

Внесем в проект изменения, касающиеся отсечения отрезков относительно области видимости. Вместо реализованного в 4-м задании алгоритма отсечения отрезков будем использовать встроенную процедуру отсечения относительно прямоугольного окна.

Откроем файл с кодом `Form1.h` и изменим процедуру отрисовки формы (`Form1_Paint`).

В этой процедуре присутствует вызов процедуры `clip` для отсечения отрезка: если процедура возвращает `true`, то отрезок (его видимая часть) отрисовывается, в противном случае отрезок пропускается. Изменим этот фрагмент программы, исключив вызов процедуры `clip`: пусть все отрезки отрисовываются без отсечения.

Теперь подключим стандартное отсечение.

В процедуре отрисовки после выполнения задания 4 должна присутствовать строка вычерчивания прямоугольника:

```
g->DrawRectangle(rectPen, Wcx, top, Wx, Wy);
```

Вместо этой строки сделаем следующее: определим прямоугольник с заданными параметрами; начертим этот прямоугольник; объявим прямоугольник областью отсечения нашего объекта `Graphics` для всех последующих операций рисования. Таким образом вместо вышеуказанной строки будет следующий код:

```
Rectangle rect = System::Drawing::Rectangle(Wcx, top, Wx, Wy);
g->DrawRectangle(rectPen, rect);
g->Clip = gcnew System::Drawing::Region(rect);
```

Для корректной отрисовки изображения приведенный код должен предшествовать циклу с вычерчиванием отрезков.

Проект можно запустить. По сравнению с заданием 4 его функциональность изменилась: наименования отрезков теперь выводятся не рядом с серединой видимой части отрезка, а рядом с точкой середины исходного отрезка.

2 Применение двойственного преобразования

Матрицу преобразования `T` будем рассматривать не как матрицу преобразования отрезков (точек концов отрезков), а как матрицу преобразования системы координат.

Прежде всего, отменим преобразование отрезков при отрисовке. То есть, вместо преобразования

```
vec A, B;
point2vec(lines[i].start, A);
point2vec(lines[i].end, B);
vec A1, B1;
timesMatVec(T,A,A1);
timesMatVec(T,B,B1);
point a, b;
vec2point(A1, a);
vec2point(B1, b);
```

с дальнейшим вычерчиванием отрезка от `a` до `b`, будем чертить отрезок от `lines[i].start` до `lines[i].end`.

Теперь применим преобразование `T` к системе координат объекта `Graphics`. Для этого воспользуемся свойством `Transform` этого объекта. Значением этого свойства является трехмерная матрица преобразования системы координат — объект типа `System::Drawing::Drawing2D::Matrix`. Предполагается, что матрица преобразования всегда имеет последнюю строку `[0 0 1]`, поэтому для инициализации объекта `Matrix` достаточно шести элементов первых двух строк матрицы преобразования. Добавим такую инициализацию ПЕРЕД вычерчиванием отрезка в цикле:

```
g->Transform = gcnew System::Drawing::Drawing2D::Matrix(T[0][0], T[1][0],
                                                        T[0][1], T[1][1],
                                                        T[0][2], T[1][2]);
```

(Можно выполнить такое присваивание только один раз перед циклом, перебирающим отрезки. Но дальнейшее выполнение задания потребует от нас дополнительного преобразования матрицы в цикле.)

Проект можно запустить. С точки зрения расположения объектов в изображении проект должен работать без изменений. Но при масштабировании изображения меняется не только положение точек, но и толщина линий отрисовки.

Подберите подходящую для вас толщину пера для вычерчивания отрезков.

3 Создание процедуры вывода экземпляра рисунка

В файле `Form1.h` в описании класса формы определите процедуру

```
private: void DrawFigure(Graphics^ g, Pen^ pen)
```

в которой пером `pen` в области рисования `g` выводится изображение вашего варианта из задания 2. Изображение должно быть вписано с сохранением пропорций в прямоугольник от -20 до 20 по оси Ox и от -30 до 30 по оси Oy (или в прямоугольник от -30 до 30 по оси Ox и от -20 до 20 по оси Oy, если ширина изображения больше его высоты).

4 Загрузка информации из файла

4.1 Формат входного файла

Предполагаем, что во входном файле могут встречаться (кроме пустых строк и строк с комментариями) строки команд следующего вида:

`frame Vcx Vcy Vx Vy` — команда выделения кадра. Если команда встречается в файле более одного раза, то каждый последующий экземпляр команды отменяет действие предыдущей;

`figure` — команда вывода экземпляра рисунка (из задания 2) с центром в начале координат;

`translate Tx Ty` — перенос начала координат в точку (`Tx`, `Ty`);

`rotate Phi` — поворот системы координат относительно начала координат против часовой стрелки на угол `Phi`, где `Phi` — угол в градусах;

`scale S` — масштабирование системы координат (увеличение единиц измерения) в `S` раз;

`pushTransform` — сохранение текущего преобразования системы координат для возможного последующего возвращения к нему;

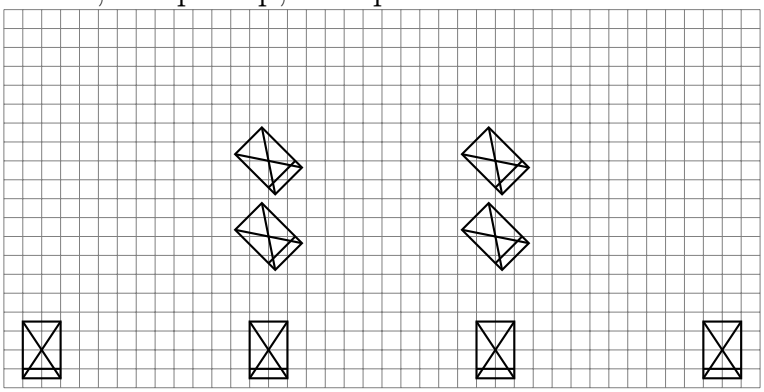
`popTransform` — извлечение из стека преобразований (с удалением из стека) матрицы преобразования и возврат (откат) к системе координат, определенной этим преобразованием.

Все команды (за исключением первой) направлены на последовательное построение изображения в правой системе координат, составленного из вариантов рисунка из задания 2. При этом рисунок всегда помещается в начало координат. Команды преобразований переноса, масштабирования, поворота относятся к преобразованиям системы координат.

Таким образом, каждой команде `figure` будет соответствовать система координат, в которой эта команда должна быть выполнена. Этой системе координат будет соответствовать матрица преобразования, которую можно получить домножая единичную матрицу последовательно на матрицы элементарных преобразований, соответствующих командам `translate`, `rotate` и `scale`.

Команды `pushTransform` и `popTransform` позволяют не повторять общие последовательности преобразований.

Так, например, изображение



может быть построено с помощью последовательности команд (если ориентация рисунка совпадает с ориентацией прямоугольника в изображении, и если одну клетку взять за 20 единиц):

```
# кадр - все представленное изображение
frame 0 0 800 400
translate 40 40
figure
# сдвиг на 12 клеток вправо (ко 2-му вертикальному ряду)
translate 240 0
figure
# запомнили позицию 1
pushTransform
translate 0 120
# запомнили позицию 2
pushTransform
rotate 45
figure
# возврат в сохраненную позицию 2 (отмена последнего поворота)
popTransform
translate 0 80
rotate 45
figure
# возврат в сохраненную позицию 1
popTransform
# сдвиг на 12 клеток вправо (к 3-му вертикальному ряду)
# и повторение предыдущей серии команд (строки 7-21)
translate 240 0
figure
pushTransform
translate 0 120
```

```
pushTransform
rotate 45
figure
popTransform
translate 0 80
rotate 45
figure
popTransform
# сдвиг на 12 клеток вправо (к 4-му вертикальному ряду)
translate 240 0
figure
```

Представленная последовательность команд не является единственно-возможной, а является лишь одним из вариантов.

4.2 Изменение формата представления векторов и матриц

Наше изображение будет состоять из экземпляров рисунка из задания 2. Для того, чтобы изобразить экземпляр рисунка будем выполнять преобразование системы координат, связанной с областью рисования, и выполнять процедуру `DrawFigure` для вывода рисунка в преобразованной системе координат. Таким образом, для каждого экземпляра рисунка необходимо хранить матрицу преобразования системы координат перед выводом рисунка.

Для обработки команд входного файла `pushTransform` и `popTransform` нам необходимо организовать стек, в который будут добавляться матрицы преобразований. Для нашего типа данных `mat`, определенного в `Transform.h` реализовать это будет относительно сложно.

Для упрощения операций со списками матриц изменим реализацию типов данных `vec` и `mat` в файле `Transform.h` и изменим процедуры обработки элементов данных этих типов.

В начале файла `Transform.h` подключим заголовочный файл

```
#include <array>
```

Вместо строк кода для определения типов `vec` и `mat`

```
typedef float vec[M];
typedef float mat[M][M];
```

добавим код

```
typedef std::tr1::array<float, M> vec;
typedef std::tr1::array<vec, M> mat;
```

Здесь вектор представляется как массив фиксированной длины `M` элементов типа `float`, а матрица представляется как массив фиксированной длины `M` векторов.

Наши процедуры, определенные в файле `Transform.cpp` будут корректно обрабатывать элементы данных такого типа, за тем исключением, что сейчас все параметры типа `mat` и `vec` передаются по значению («значением» теперь является весь вектор или вся матрица, а не указатель, как было в случае использования стандартных массивов). Для того, чтобы процедуры возвращали матрицу или вектор обновленного типа нужно указать для выходных параметров, что эти параметры должны передаваться по ссылке. Например, для процедуры `times` изменить описание параметра `c` следующим образом:

```
void times(mat a, mat b, mat &c)
```

Такие изменения нужно внести в заголовок каждой процедуры, для каждого выходного параметра типа `vec` или `mat`. Изменения необходимо продублировать для заголовков процедур в файле `Transform.h`.

Подключение заголовочного файла

```
#include <array>
```

необходимо добавить в основной файл проекта — ⟨*имя проекта*⟩`.cpp`. Здесь его добавим ПЕРЕД подключением заголовочного файла `Transform.h`.

Для проверки корректности внесенных изменений можно запустить проект. Работа программы не должны претерпеть каких либо изменений.

4.3 Список матриц преобразований

Список отрезков `lines` в настоящем задании не понадобится. Вместо него необходимо организовать хранение списка матриц преобразований. Будем хранить такой список в глобальной переменной `matrices`, объявление которой сделаем в файле `Transform.h`, а инициализацию проведем в файле `Transform.cpp`.

В файле `Transform.h` подключим заголовочный файл

```
#include <vector>
```

После объявления переменной `T` добавим объявление переменной `matrices`:

```
extern std::vector<mat> matrices;
```

В файле `Transform.cpp` после инициализации переменной `T` добавим описание:

```
std::vector<mat> matrices(0);
```

Подключение заголовочного файла

```
#include <vector>
```

необходимо добавить в основной файл проекта — ⟨*имя проекта*⟩`.cpp`. Здесь его добавим ПЕРЕД подключением заголовочного файла `Transform.h`.

4.4 Процедура отрисовки набора рисунков

Внесем изменения в процедуру отрисовки изображения (`Form1_Paint`). Вместо цикла по списку отрезков организуем цикл по списку матриц

```
for (int i = 0; i < matrices.size(); i++) {
    ...
}
```

В цикле каждую матрицу в списке будем умножать на матрицу преобразования `T` для получения матрицы общего преобразования системы координат. На основе этой матрицы будем создавать объект `Matrix` для свойства `Transform` графической области:

```
mat C;
times(T, matrices[i], C);
g->Transform = gcnew System::Drawing::Drawing2D::Matrix(C[0][0], C[1][0],
                                                        C[0][1], C[1][1],
                                                        C[0][2], C[1][2]);
```

После этого вызываем процедуру `DrawFigure` для вывода соответствующего экземпляра рисунка.

4.5 Чтение входного файла

При чтении данных из файла нам потребуется работать со стеком матриц. Для организации стека подключим заголовочный файл

```
#include <stack>
```

в основном файле проекта — ⟨*имя проекта*⟩`.cpp` ПЕРЕД подключением заголовочного файла `Transform.h`.

Внесем изменения в процедуру чтения входного файла (`btnOpen_Click`). Чтение файла должно приводить к изменению матрицы `T` и списка матриц `matrices`.

В случае успешного открытия файла вместо очистки списка отрезков

```
lines.Clear();
```

очистим список `matrices`:

```
matrices.clear();
```

и объявим стек матриц `matStack`:

```
std::stack<mat> matStack;
```

Кроме того, нам понадобится матрица, в которой будем накапливать преобразование для системы координат, в соответствии с командами из входного файла:

```
mat K; unit(K);
```

Как и раньше, матрица `T` должна здесь инициализироваться единичной матрицей:

```
unit(T);
```

Дальнейшие изменения должны коснуться цикла чтения строк файла. Каждая значащая строка файла начинается с имени команды. Поэтому, сразу после объявления строкового потока `s` объявим строковую переменную `cmd` и прочитаем ее значение из потока:

```
std::string cmd;
s >> cmd;
```

Последующие действия должны зависеть от того, какая команда была прочитана:

```
if ( cmd == "frame" ) {
    ...
}
else if ( cmd == "figure" ) {
    ...
}
else if ( cmd == "pushTransform" ) {
    ...
}
else if ( cmd == "popTransform" ) {
    ...
}
else if ( cmd == "translate" ) {
    ...
}
else if ( cmd == "scale" ) {
    ...
}
else if ( cmd == "rotate" ) {
    ...
}
```

В случае, если прочитана команда `frame`, необходимо прочитать из потока 4 параметра и провести инициализацию матрицы `T` с помощью процедуры `frame`.

Если прочитана команда `figure`, то матрица преобразования `K` — матрица преобразования системы координат перед отрисовкой очередного экземпляра рисунка. То есть копию матрицы `K` нужно сохранить в списке `matrices`:

```
matrices.push_back(K);
```

Выполнение команды `pushTransform` заключается в записи текущей матрицы `K` в стек

```
matStack.push_back(K);
```

а выполнение команды `popTransform` заключается в обновлении матрицы `K`, при котором ее значение берется из стека:

```
K = matStack.top();
matStack.pop();
```

Остальные три команды являются командами преобразования системы координат. Выполнение каждой такой команды заключается в домножении матрицы `K` на матрицу соответствующего преобразования. Например, для команды `rotate`:

```
1 float Phi;
2 s >> Phi;
3 float pi = 3.1415926535;
4 float PhiR = Phi * (pi / 180);
5 mat C, C1;
6 rotate(PhiR, C);
7 times(K, C, C1);
8 K = C1;
```

Строки 3 и 4 приведенного кода переводят значение угла поворота из градусов в радианы. Обратите внимание на строку 7: так как матрица преобразования `K` — матрица, совмещающая последовательность преобразований системы координат, домножение этой матрицы на матрицу очередного преобразования проводится справа.