Появление структурного подхода, основные принципы

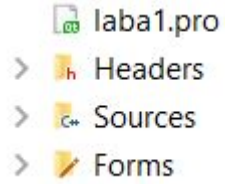
- IBM 1961 г
- В основе декомпозиция, разбиение задачи на подзадачи
- Выделение не более 7 подзадач
- Глубина вложенности конструкции не более 3
- Размер подзадачи < 200 строк
- Количество передаваемых на модуль данных не более 3
- Все данные передаются явно через список параметров
- Структурирование данных - структура данных должна соответствовать уровням абстракции
- Проектирование, кодирование, тестирование
- Логика должна концентрироваться на более высоких уровнях абстракций

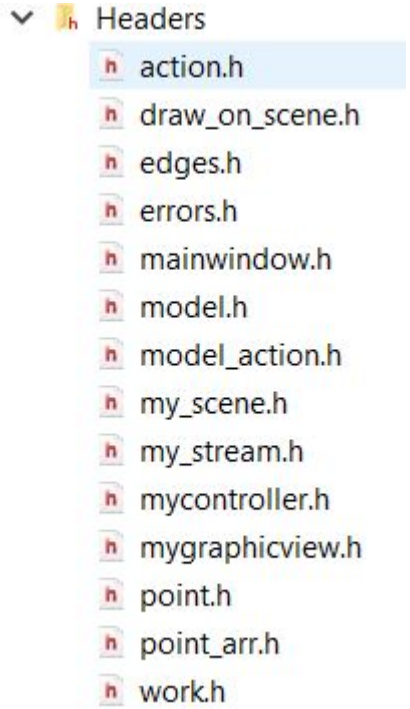
Появление структурного подхода, основные принципы

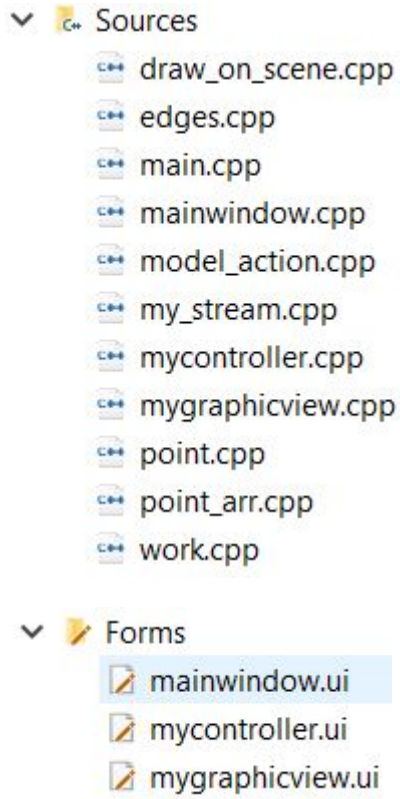
- Реализация алгоритма любой сложности при помощи трех базовых логических структур: условие, следование, цикл
- Размер рабочей группы не должен превышать 7
- Уровень руководителя должен быть выше, чем у подчиненного

1. Проектирование

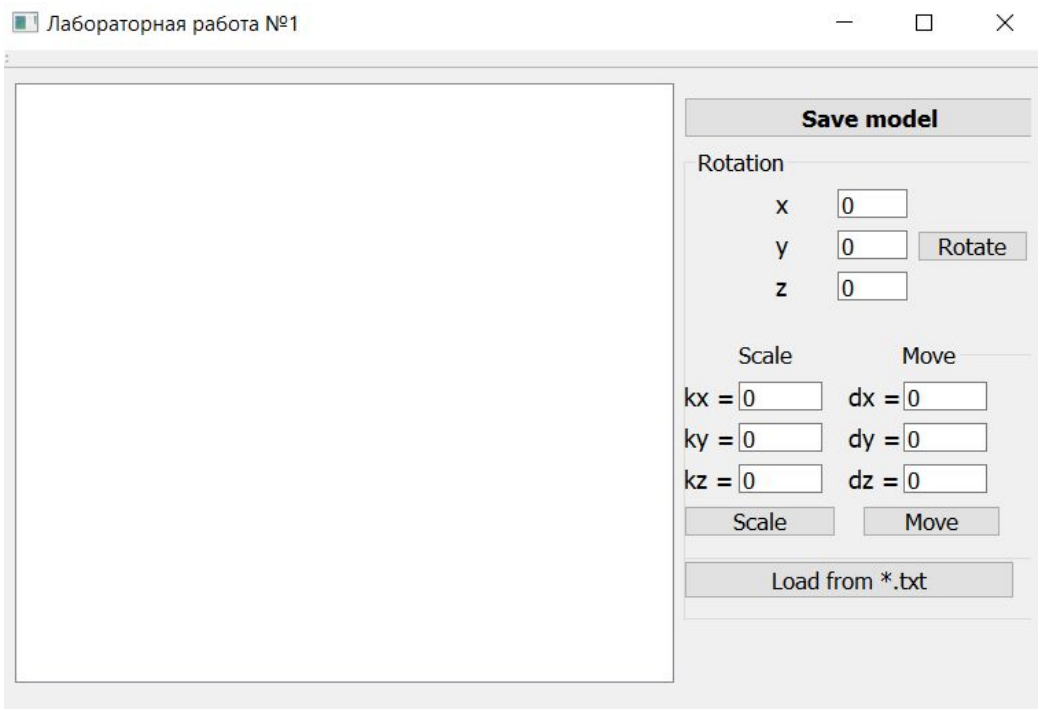
Глобально все составляющие проекта можно будет поделить на 3 части - файлы ui - формы, отвечающие за интерфейс, заголовочные файлы и исходники.







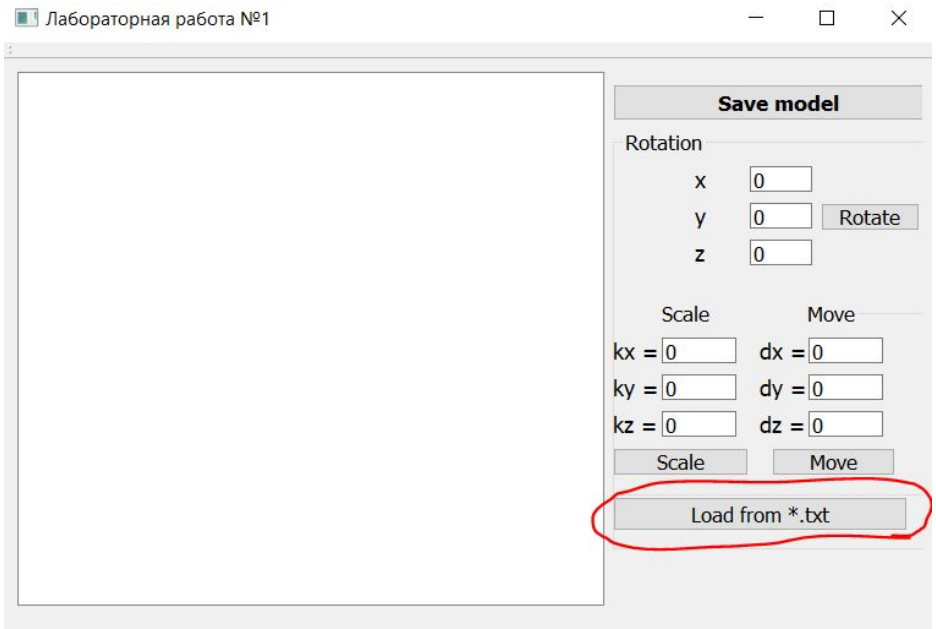
1. Проектирование. Интерфейс.



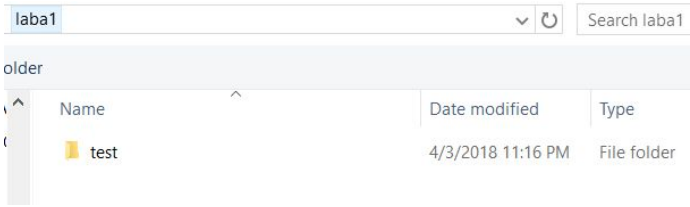
Интерфейс программы состоит из трех частей:

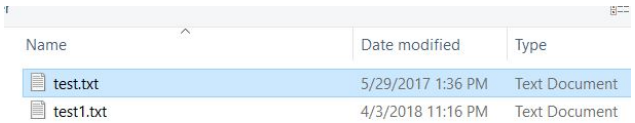
1. окно отображения - mainwindow.ui
2. окно отрисовки модели - mygraphicview.ui
3. панель управления - mycontroller.ui

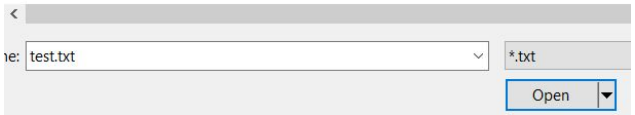
1. Проектирование. Определение точки входа.



У программы одна точка входа - загрузка данных из текстового файла







1. Проектирование.

Работа с входным потоком и выходным. Входной поток - считывание данных из файла, выходной поток - запись текущих координат вершин в файл, по требованию пользователя.

```
struct IN_Stream {
    std::ifstream inp;
};

struct OUT_Stream {
    std::ofstream out;
};
```

```
int Open_Stream(IN_Stream &stream, const char* filename);
int Read_Stream(double &x, IN_Stream &stream);
int Close_Stream(IN_Stream &stream);

int Open_Stream(OUT_Stream &stream, const char* filename);
int Print_Stream(OUT_Stream &stream, const char* str);
int Close_Stream(OUT_Stream &stream);
```

1. Проектирование. Определение точки входа.

```
struct vertex_arr {
    Point *arr = NULL;
    int N_v = 0;
};
```

```
struct Point {
    double x;
    double y;
    double z;
};
```

У программы будет возможность - записи текущего состояния вершин в файл.

Данные из текстового файла представляют собой набор троек точек в Декартовой системе координат.Это можно выделить как уровень абстракции.

```
//загрузка массива точек опр. длины из файла
int Load_point_arr(vertex_arr &vertex, IN_Stream &stream);

int Allocate_Point_arr(vertex_arr &vertex);
int Free_Point_arr(vertex_arr &vertex);

//запись массива точек в файл
int Save_point_arr(const vertex_arr &vertex, OUT_Stream &stream);

int Get_N_vertex(const vertex_arr &vertex);

Point Get_Point(const vertex_arr &vertex, int i);

Point* Get_vertex_arr(const vertex_arr &vertex);
```

1. Проектирование.

```
//возможные ошибки работы функций
enum ERRORS { OK = 0,
              FILE_NOT_FIND,
              FILE_ERROR,
              MEMORY_ERROR,
              MODEL_EMPTY,
              SCENE_NOT_FOUND
            };
```

OK - код ошибки, который означает безошибочную работу функции

FILE_NOT_FOUND - ошибки ненайденного файла

FILE_ERROR - ошибка для неверного формата файла, или данных неверно записанных в файле (не по формату)

MEMORY_ERROR - ошибка памяти, при выделении для хранения структур данных

MODEL_EMPTY - задана пустая модель

SCENE_NOT_FOUND - неверно проинициализирована или не найдена сцена для отрисовки

1. Проектирование.

Определение структуры хранения модели.

```
struct Model {
    vertex_arr vertex;
    edges_arr edges;
};
```

1. Проектирование.

Следующий уровень - формирование длин сторон фигуры. Работа с гранями каркасной модели.

```
typedef int t_edge[2];

struct edges_arr {
    t_edge *arr = NULL;
    int N_e = 0;
};

edges_arr Init_edges();
```

```
edges_arr Init_edges();

int Load_edge_arr(edges_arr &edges, IN_Stream &stream, int max_vertex);
int Load_edge(t_edge& p, IN_Stream &stream, int max_vertex);

int Allocate_Edge_arr(edges_arr &edges);
int Free_Edge_arr(edges_arr &edges);

int Save_edge_arr(const edges_arr &edges, OUT_Stream &stream);
int Save_edge(const t_edge& p, OUT_Stream &stream);

int Get_N_edges(const edges_arr &edges);

int Get_edge_start(const edges_arr &edges, int i);
int Get_edge_end(const edges_arr &edges, int i);

t_edge* Get_edges_arr(const edges_arr &edges);
```

1. Проектирование.

Работа с моделью

```
Model Init_model();
int Is_init_model(const Model &model);

//преобразование модели
int Rotate_model(Model &model, const Rotate &act);
int Scale_model(Model &model, const Scale &act);
int Move_model(Model &model, const Move &act);

//загрузка модели из файла
int LoadModel(Model &model, const Create &act);

//сохранение модели в файл
int SaveModel(const Model &model, const Create &act);

//освобождения массивов вершин и ребер
int Free_model(Model &model);

int Draw_model(My_Scene &scene, const Model &model);

int Get_N_vertex(const Model &model);

int Get_N_edges(const Model &model);

vertex_arr Get_vertex_arr(const Model &model);
//Point* Get_vertex_arr(const Model &model);
edges_arr Get_edges_arr(const Model &model);
//t_edge* Get_edges_arr(const Model &model);
```

1. Проектирование.

```
struct Point {
    double x;
    double y;
    double z;
};

//тип для хранения матриц преобразование в N-мерном пр-ве
typedef double t_matrix[N_DIMEN][N_DIMEN];

//тип для хранения вектора-точки в N-мерном пр-ве
typedef double t_vect[N_DIMEN];

//получние результирующей матрицы поворота по 3-м углам
void GetResultMatrix(t_matrix a, const Rotate &act);

void GetResultMatrix(t_matrix a, const Scale &act);
void GetResultMatrix(t_matrix a, const Move &act);

//конвертация точки в вектор
int From_vec_to_Point(Point &p, const t_vect &vec);
int From_Point_to_vec(t_vect &vec, const Point &p);

//Применение матрицы преобразований к одной точке
int Change_Point_with_matrix(Point &p, const t_matrix &m_rotate);

//загрузка точки опр. длины из файла
int Load_point(Point& p, IN_Stream &stream);

//запись точки в файл
int Save_point(const Point& p, OUT_Stream &stream);

int Draw_line(My_Scene &scene, const Point &a, const Point &b);
```

Работа с точками, выполнение необходимых преобразований.

1. Проектирование.

Уровень контроллера действий

```
enum type_action { ROTATE, CREATE, MOVE, SCALE, FREE, SAVE, DRAW };
union t_action {
    Rotate rotat;
    Scale scal;
    Create creat;
    Move mov;
    bool free;
};
// принимает на вход модель, действие (одно из описанных)
// и тип действия
int main_controller(My_Scene &scene, const t_action &act, type_action t);
```

1. Проектирование.

Выделяем структуры действий с моделью.

```
//структуры действий
struct Rotate {
    double x_angle;
    double y_angle;
    double z_angle;
};
struct Scale {
    double kx;
    double ky;
    double kz;
};
struct Move {
    double dx;
    double dy;
    double dz;
};
struct Create {
    char fileName[FILE_NAME_SIZE];
};
```

1. Проектирование.

Далее переходим на уровень интерфейса. Работа с формированием модели закончена и следует приступить к ее отрисовке в исходном виде.

```
struct My_Scene {
    QGraphicsScene *scene = NULL;
    double x_center;
    double y_center;
};
```

Выделяем еще один уровень - структуру для сцены. Это самостоятельный “уровень абстракции, который будет использоваться в течении всей работы программы”

Создаем две функции, которые будут отвечать за отрисовку модели и очищение сцены.
То есть - работа со сценой и моделью.

```
int Draw_2d_line(My_Scene &scene, double x1, double y1,
                 double x2, double y2);

int Clean_Scene(My_Scene &scene);
```

1. Проектирование.

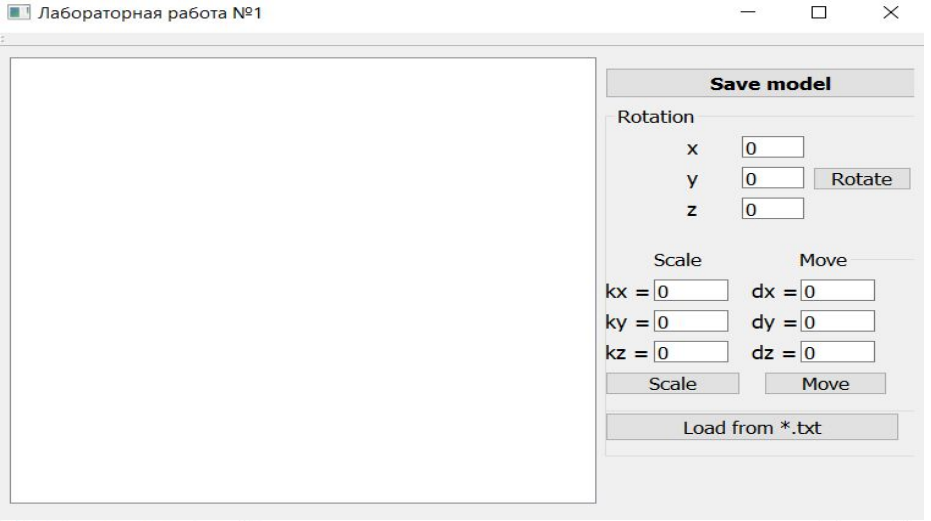
Использование классов на уровне интерфейса.

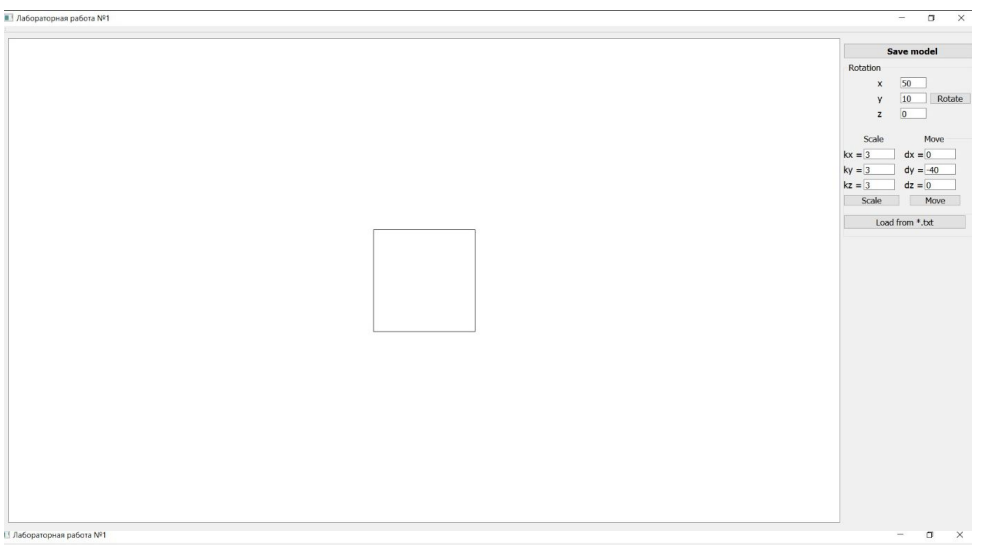
```
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void SendingScene(My_Scene *my_scene);
private:
    Ui::MainWindow  *ui;
    MyGraphicView   *myPicture;     // Наш кастомный виджет
    MyController    *myController;
};
```

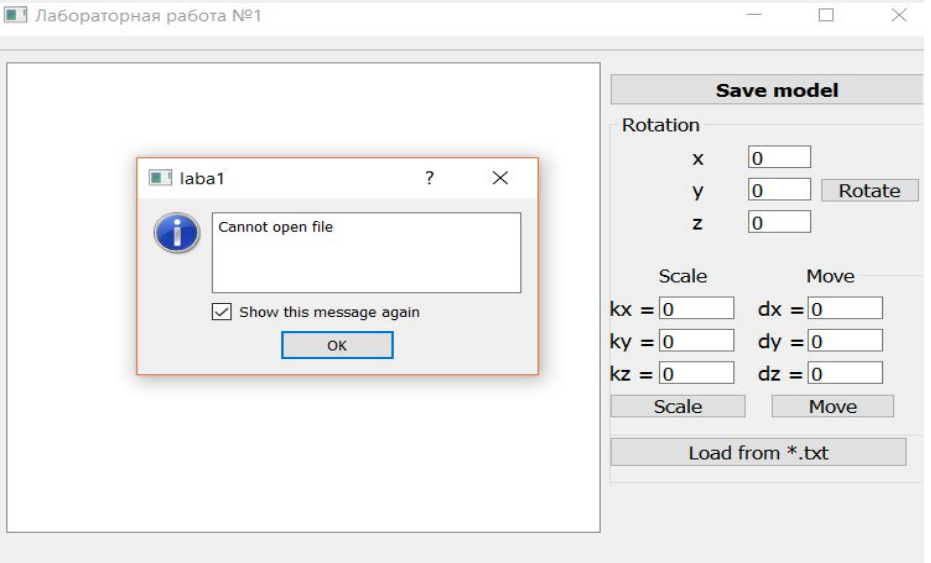
```
#define POINT_SIZE 3

namespace Ui {
class MyGraphicView;
}

class MyGraphicView : public QGraphicsView
{
    Q_OBJECT

public:
    explicit MyGraphicView(QWidget *parent = 0);
    ~MyGraphicView();
    void Paint(Model &mod);
    void Paint(QGraphicsScene *scene);
    void Connect();
signals:
    void SendScene(My_Scene* my_scene);
private:
    Ui::MyGraphicView *ui;
    My_Scene my_scene;
};
```

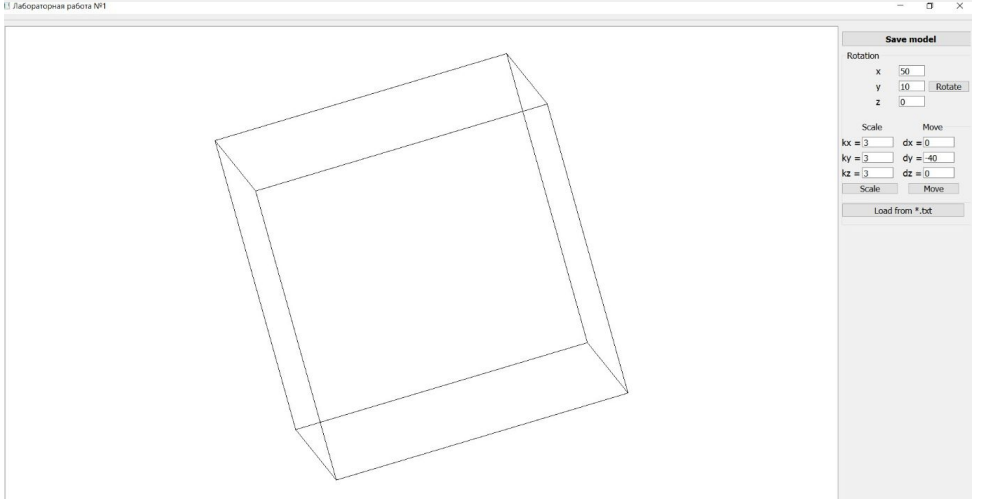
```
using namespace std;
enum Text_Error { EMPTY, E_SYMBOL, NO_ER };

namespace Ui {
class MyController;
}
```

```
class MyController : public QWidget
{
    Q_OBJECT

public:
    explicit MyController(QWidget *parent = 0);
    ~MyController();
    void GetScene(My_Scene *scene);
signals:
    void AnswerChange(Model &mod);
    void SceneChange(QGraphicsScene *scene);
private slots:
    void on_rotateButton_clicked();

    void on_scaleButton_clicked();

    void on_moveButton_clicked();

    void on_fileButton_clicked();

    void on_saveButton_clicked();

private:

    double *GetData(vector <QLineEdit*> &vec);
    My_Scene scene;
    Ui::MyController *ui;
    Model model;
    QWidget *par;
    QRegExpValidator *Validator;
};
```


Лабораторная работа №1
Save model
Rotation
x
0
y
0
Rotate
z
0
Scale
Move
kx =
0
dx =
0
ky =
0
dy =
0
kz =
0
dz =
0
Scale
Move
Load from *.txt


Лабораторная работа №1
Save model
Rotation
x
50
y
10
Rotate
z
0
Scale
Move
kx =
3
dx =
0
ky =
3
dy =
-40
kz =
3
dz =
0
Scale
Move
Load from *.txt


Лабораторная работа №1
laba1
?
Cannot open file
Show this message again
OK
Save model
Rotation
x
0
y
0
Rotate
z
0
Scale
Move
kx =
0
dx =
0
ky =
0
dy =
0
kz =
0
dz =
0
Scale
Move
Load from *.txt


Лабораторная работа №1
Save model
Rotation
x
50
y
10
Rotate
z
0
Scale
Move
kx =
3
dx =
0
ky =
3
dy =
-40
kz =
3
dz =
0
Scale
Move
Load from *.txt