```cpp
#include "StdAfx.h"
#include "SMPConditions.h"

extern int LibMessage(LPCTSTR str, int flags = MB_OK);

void SMPConditions::createModel() {
    this->addEmbodiment();
    this->drawBase();
    this->addExtrusion();
    this->addRounding();
    this->addBladePlanes();
    this->addHole();
}

bool SMPConditions::updateParameters() {
    double angle_main = 0;
    this->points.clear();
    if (this->isCircle()) {
        this->updateParametersCircle();
    }
    else if (this->isNGon()) {
        this->updateParametersNGon(&angle_main);
    }
    else if (this->isTrigon()) {
        this->updateParametersTrigon(&angle_main);
    }
    else if (this->isRhombus()) {
        this->updateParametersRhombus(&angle_main);
    }
    if (this->hasHole && (this->holeRadius > this->innerRadius)) {
        this->holeRadius = this->innerRadius;
    }
    return true;
}

void SMPConditions::updateParametersCircle() {
    if (!this->isCircle()) {
        return;
    }
    this->innerRadius = this->size;
    this->outerRadius = this->innerRadius;
    this->side = this->innerRadius;
}

void SMPConditions::updateParametersNGon(double* angle_main) {
    if (!this->isNGon()) {
        return;
    }
    unsigned short count = this->getNGonSideCount();
    double angle_vertex = 360.0 / count;
    double angle_rad = (180.0 - angle_vertex) / 2;
    *angle_main = angle_rad;
    if (this->isByInnerRadius()) {
        this->innerRadius = this->size;
    }
    else if (this->isByOuterRadius()) {
        this->innerRadius = this->size * sind(angle_rad);
    }
    else if (this->isBySide()) {
        this->innerRadius = this->size * sind(angle_rad) * sind(angle_rad) / sind(angle_vertex);
    }
    this->outerRadius = this->innerRadius / sind(angle_rad);
    this->delta_max = this->outerRadius;
    if (this->isByOuterRadius()) {
        double delta_rad = this->roundingRadius / sind(angle_rad) - this->roundingRadius;
        this->outerRadius += delta_rad;
        this->innerRadius = this->outerRadius * sind(angle_rad);
    }
    this->side = this->outerRadius * sind(angle_vertex / 2) / sind(angle_rad);
    if (!this->isByOuterRadius()) {
        double delta_rad = this->roundingRadius / sind(angle_rad) - this->roundingRadius;
        this->outerRadius -= delta_rad;
    }
}

void SMPConditions::updateParametersTrigon(double* angle_main) {
    if (!this->isTrigon()) {
        return;
    }
    double angle_vertex = 60;
    double angle_greater = 80;
    double angle_less = 40;
    *angle_main = angle_less;
    if (this->isByInnerRadius()) {
        this->innerRadius = this->size;
    }
    else if (this->isByOuterRadius()) {
        this->innerRadius = this->size * sind(angle_less);
    }
    else if (this->isBySide()) {
        this->innerRadius = this->size * sind(angle_less) * sind(angle_greater) / sind(angle_vertex);
    }
    this->outerRadius = this->innerRadius / sind(angle_less);
    this->delta_max = this->outerRadius;
    if (this->isByOuterRadius()) {
        double delta_rad = this->roundingRadius / sind(angle_less) - this->roundingRadius;
        this->outerRadius += delta_rad;
        this->innerRadius = this->outerRadius * sind(angle_less);
    }
    this->side = this->outerRadius * sind(angle_vertex) / sind(angle_greater);
    if (!this->isByOuterRadius()) {
        double delta_rad = this->roundingRadius / sind(angle_less) - this->roundingRadius;
        this->outerRadius -= delta_rad;

```

```cpp
}

void SMPConditions::updateParametersRhombus(double* angle_main) {
    if (!this->isRhombus()) {
        return;
    }
    double angle = this->getRhombusAngle() / 2;
    *angle_main = angle;
    if (this->isByInnerRadius()) {
        this->innerRadius = this->size;
    }
    else if (this->isByOuterRadius()) {
        this->innerRadius = this->size * sind(angle);
    }
    else if (this->isBySide()) {
        this->innerRadius = this->size * sind(angle) * cosd(angle);
    }
    this->outerRadius = this->innerRadius / sind(angle);
    this->delta_max = this->outerRadius;
    if (this->isByOuterRadius()) {
        double delta_rad = this->roundingRadius / sind(angle) - this->roundingRadius;
        this->outerRadius += delta_rad;
        this->innerRadius = this->outerRadius * sind(angle);
    }
    this->side = this->outerRadius / cosd(angle);
    if (!this->isByOuterRadius()) {
        double delta_rad = this->roundingRadius / sind(angle) - this->roundingRadius;
        this->outerRadius -= delta_rad;
    }
}

void SMPConditions::addBladePlanes() {
    if (this->isCircle()) {
        return;
    }
    int count = 0;
    double angle_vertex = 0;
    double angle_main = 0;
    if (this->isNGon()) {
        count = this->getNGonSideCount();
        angle_vertex = 360.0 / count;
        angle_main = (180.0 - angle_vertex) / 2;
    }
    else if (this->isTrigon()) {
        count = 3;
        angle_vertex = 120;
        angle_main = 40;
    }
    else if (this->isRhombus()) {
        count = 2;
        angle_vertex = 180;
        angle_main = this->getRhombusAngle() / 2;
    }

    double delta = this->delta_max - this->outerRadius;
    double max_x = this->roundingRadius * cosd(angle_main);
    double max_y = -(this->roundingRadius - this->roundingRadius * cosd(angle_main));
    double s_x, s_y, t_x, t_y;
    s_x = this->parameterS / 2;
    t_y = -this->parameterT;

    if (s_x <= max_x) {
        double cat2 = cat(this->roundingRadius, s_x);
        s_y = cat2 - this->roundingRadius;
    }
    else {
        s_y = delta - s_x * cotd(angle_main);
    }
    if (t_y >= max_y) {
        double cat1 = (this->roundingRadius - this->parameterT);
        t_x = -cat(this->roundingRadius, cat1);
    }
    else {
        t_x = -(delta - t_y) * tand(angle_main);
    }

    double length_x = s_x - t_x;
    double length_y = s_y - t_y;
    double direction_angle = atand(length_y / length_x);
    if (this->angleEtaAuto) {
        this->angleEta = direction_angle + 90;
    }
    else {
        direction_angle = this->angleEta - 90;
    }

    double x = this->coordinates.X;
    double y = this->coordinates.Y;
    double rotation_angle;
    if (this->isNGon()) {
        double begin_angle = atand(t_y / t_x);
        double begin_length = hyp(t_x, t_y);
        double offset_x = begin_length * cosd(angle_main + begin_angle);
        double offset_y = -begin_length * sind(angle_main + begin_angle);
        x += this->outerRadius * cosd(angle_vertex / 2) + offset_x;
        y += this->outerRadius * sind(angle_vertex / 2) + offset_y;
        rotation_angle = 90 - direction_angle + angle_vertex / 2;
    }
    else {
        x += -t_x;
        y += this->outerRadius + t_y;
        rotation_angle = 180 - direction_angle;
    }

    this->addBladePlane(count, x, y, rotation_angle);
}

```

```cpp
void SMPConditions::addBladePlane(int count, double x, double y, double rotation_angle) {
    double z = this->coordinates.Z;
    double nutation_angle;
    double precession_angle = 0;
    double x_c, y_c;
    if (rotation_angle > 180) {
        nutation_angle = -this->angleLambdaD;
        x_c = 0;
        y_c = this->cutRadius;
    }
    else {
        nutation_angle = this->angleLambdaD;
        x_c = this->cutRadius;
        y_c = 0;
    }

    IAuxiliaryGeomContainerPtr geom_container(this->part7);
    ILocalCoordinateSystemPtr local_cs = geom_container->LocalCoordinateSystems->Add();
    local_cs->Name = _T("ЛСК режущей кромки");
    local_cs->X = x;
    local_cs->Y = y;
    local_cs->Z = z;
    local_cs->OrientationType = ksOrientationTypeEnum::ksEulerCorners;
    ILocalCSEulerParamPtr local_cs_parameters = local_cs->GetLocalCSParameters();
    local_cs_parameters->NutationAngle = nutation_angle;
    local_cs_parameters->PrecessionAngle = precession_angle;
    local_cs_parameters->RotationAngle = rotation_angle;
    local_cs->Update();

    ISketchPtr sketch = this->model_container->Sketchs->Add();
    sketch->CoordinateSystem = local_cs;
    sketch->Plane = local_cs->GetDefaultObject(ksObj3dTypeEnum::o3d_planeYOZ);
    sketch->Name = _T("Профиль режущей кромки");

    IFragmentDocumentPtr sketch_document = sketch->BeginEdit();
    IDrawingContainerPtr drawing_container = this->getDrawingContainer(sketch_document);
    ICirclePtr figure = drawing_container->Circles->Add();
    figure->Xc = x_c;
    figure->Yc = y_c;
    figure->Radius = this->cutRadius;
    figure->Update();
    sketch->EndEdit();

    ICutExtrusionPtr extrusion = this->model_container->Extrusions->Add(ksObj3dTypeEnum::o3d_cutExtrusion);
    extrusion->Name = _T("Элемент выдавливания режущей кромки");
    extrusion->Direction = ksDirectionTypeEnum::dtBoth;
    extrusion->SetSideParameters(
        false, ksEndTypeEnum::etThroughAll,
        0, 0, true, NULL
    );
    extrusion->SetSideParameters(
        true, ksEndTypeEnum::etThroughAll,
        0, 0, true, NULL
    );
    extrusion->Sketch = sketch;
    extrusion->Update();

    IPoints3DPtr points = this->model_container->Points3D;
    IPoint3DPtr point1 = points->Add();
    IPoint3DPtr point2 = points->Add();

    point1->Name = _T("Точка 1 оси массива режущих кромок");
    point1->X = this->coordinates.X;
    point1->Y = this->coordinates.Y;
    point1->Z = this->coordinates.Z;
    point1->Update();

    point2->Name = _T("Точка 2 оси массива режущих кромок");
    point2->X = this->coordinates.X;
    point2->Y = this->coordinates.Y;
    point2->Z = this->coordinates.Z + this->getHeight();
    point2->Update();

    IAxis3DBy2PointsPtr axis = this->model_container->AddObject(ksObj3dTypeEnum::o3d_axis2Points);
    axis->Name = _T("Ось массива режущих кромок");
    axis->Point1 = point1;
    axis->Point2 = point2;
    axis->Update();

    ICircularPatternPtr circular_pattern = this->model_container->FeaturePatterns->Add(ksObj3dTypeEnum::o3d_circularCopy);
    circular_pattern->Name = _T("Массив режущих кромок");
    circular_pattern->InitialObjects = (ICutExtrusion*)extrusion;
    circular_pattern->BuildingType = ksCircularPatternBuildingTypeEnum::ksCPSaveAll;
    circular_pattern->BasePoint = this->part7->GetDefaultObject(ksObj3dTypeEnum::o3d_pointCS);
    circular_pattern->Axis = axis;
    circular_pattern->Step2 = 360;
    circular_pattern->Count1 = 1;
    circular_pattern->Count2 = count;
    circular_pattern->BoundaryInstancesStepFactor2 = true;
    circular_pattern->Update();
}
```