```cpp
#ifndef TRIANGLEREFINER_HPP
#define TRIANGLEREFINER_HPP

#include "GenericDomain.hpp"
#include "GenericMesh.hpp"
#include "Eigen/Eigen"

using namespace std;
using namespace Eigen;

namespace GeDiM
{
class TriangleRefiner
{
private:
        /*
        Seguono dichiarazione e definizione di funzioni semplici o di supporto
        */

        // Ritorna la distanza tra due punti
        static double SqrDistance(const GenericPoint *P1, const GenericPoint *P2)
        {
                return (P1->Coordinates() - P2->Coordinates()).squaredNorm();
        }

        // Ritorna il lato più lungo di una cella
        const GenericEdge *LongestEdge(const GenericCell *C)
        {
                double a = SqrDistance(C->Edge(0)->Point(0), C->Edge(0)->Point(1));
                double b = SqrDistance(C->Edge(1)->Point(0), C->Edge(1)->Point(1));
                double c = SqrDistance(C->Edge(2)->Point(0), C->Edge(2)->Point(1));
                if ((a >= b) and (a >= c))
                        return C->Edge(0);
                else if ((b >= a) and (b >= c))
                        return C->Edge(1);
                else
                        return C->Edge(2);
        }

        // Ritorna true se il triangolo ha almeno un lato marcato
        bool HasMarkedEdges(const GenericCell *C)
        {
                return (idEdgesToCut.at(C->Edge(0)->Id())) or
(idEdgesToCut.at(C->Edge(1)->Id())) or (idEdgesToCut.at(C->Edge(2)->Id()));
        }

        // Ritorna true se il lato è sul bordo ovvero confina con una cella NULL
        bool IsOnBorder(const GenericEdge *E)
        {
                return (E->Cell(0) == NULL) or (E->Cell(1) == NULL);
        }

```

```
52          // Creano in modo corretto e coerente vari tipi di oggetti geometrici
53          GenericPoint *NewPoint()
54          {
55                  GenericPoint *P = meshPointer->CreatePoint();
56                  meshPointer->AddPoint(P);
57                  return P;
58          }
59          GenericEdge *NewEdge()
60          {
61                  GenericEdge *E = meshPointer->CreateEdge();
62                  meshPointer->AddEdge(E);
63                  idEdgesToCut.push_back(false); //Creando un lato è necessario
estendere idEdgesToCut perchè lo comprenda
64                  return E;
65          }
66          GenericCell *NewCell()
67          {
68                  GenericCell *C = meshPointer->CreateCell();
69                  meshPointer->AddCell(C);
70                  return C;
71          }
72
73          // Riempie un lato vuoto con informazioni geometriche di punti e celle
74          void SetEdgeGeometry(GenericEdge *E, const GenericPoint *P0, const
GenericPoint *P1, const GenericCell *right, const GenericCell *left)
75          {
76                  E->AddPoint(P0);
77                  E->AddPoint(P1);
78                  E->InitializeCells(2);
79                  E->AddCell(right); // RightCell = cell[0]
80                  E->AddCell(left);  // LeftCell = cell[1]
81          }
82
83          // Inizializza le informazioni di parentela fra un padre e due figli
84          void SetFamily(GenericTreeNode *father, GenericTreeNode *C1, GenericTreeNode
*C2)
85          {
86                  father->InitializeChilds(2);
87                  father->AddChild(C1);
88                  father->AddChild(C2);
89                  C1->SetFather(father);
90                  C2->SetFather(father);
91          }
92
93          // Inizializzano le informazioni geometriche su punti e lati di una cella
vuota
94          void SetCellPoints(GenericCell *C, const GenericPoint *P0, const
GenericPoint *P1, const GenericPoint *P2)
95          {
96                  C->InitializePoints(3);
97                  C->AddPoint(P0);
98                  C->AddPoint(P1);
99                  C->AddPoint(P2);
```

```
100         }
101         void SetCellEdges(GenericCell *C, const GenericEdge *E0, const GenericEdge
    *E1, const GenericEdge *E2)
102         {
103                 C->InitializeEdges(3);
104                 C->AddEdge(E0);
105                 C->AddEdge(E1);
106                 C->AddEdge(E2);
107         }
108
109         // Ritorna true se nel vettore idEdgesToCut c'è ancora almeno un true
110         bool anyTriangleToCut()
111         {
112                 for (unsigned edgeId = 0; edgeId < meshPointer->NumberOfEdges();
    edgeId++)
113                         if (idEdgesToCut.at(edgeId))
114                                 return true;
115
116                 return false;
117         }
118
119         /*
120                 Seguono dichiarazioni di funzioni algoritmiche importanti o
    complesse
121         */
122         void RotateCell(GenericCell
    *C);                                                    // Riordina i puntatori a
    lati e punti del triangolo in modo coerente
123         void PensaciTuAlLatoIgnoto(GenericCell *C, GenericEdge *E);   // Sistema le
    condizioni di vicinanza tra un triangolo ed un lato "esterno"
124         void RefinePairedTriangles(GenericCell *C0, GenericCell *C1); // Taglia
    contemporanemante due triangoli che condividono il lato lungo
125         void RefineBorderTriangle(GenericCell
    *C0);                                       // Taglia un triangolo che ha il lato
    lungo sul bordo della mesh
126
127         /*
128                 Membri della classe TriangleRefiner
129         */
130
131         GenericMesh *meshPointer;  // Puntatore alla mesh
132         vector<bool> idEdgesToCut; // Vettore di variabili booleane. Se l'elemento
    n-esimo del vettore è true allora il lato con id = n è da tagliare
133
134 public:
135         // Nel costruttore richiedo una mesh e preparo il vettore di bool grande
    abbastanza
136         TriangleRefiner(GenericMesh &mesh)
137         {
138                 meshPointer = &mesh;
139                 idEdgesToCut.assign(meshPointer->NumberOfEdges(), false);
140         }
141         ~TriangleRefiner()
142         {
```

```
143                 meshPointer = NULL;
144                 idEdgesToCut.clear();
145         }
146 
147         // Funzione che ruota la cella e marca da tagliare il lato più lungo
148         void PrepareTriangle(const unsigned int &value);
149 
150         // Funzione che ricalcola tutte le informazioni di vicinanza dei punti
151         void AggiornaInformazioniPunti();
152 
153         // Raffina la mesh in maniera conforme
154         Output::ExitCodes RefineMesh();
155 };
156 } // namespace GeDiM
157 
158 #endif
159 
```