```cpp
//  g++ -O3 -Wall -W -Wfloat-equal -Wshadow -Wredundant-decls -Weffc++ -pedantic main.cpp polygon.cpp
//  clang++ -O3 -Weverything -Wno-conversion -Wno-c++98-compat -Wno-padded main.cpp polygon.cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <iterator>
#include <algorithm>   // sort
#include <cstdlib>     // srand, rand
#include <ctime>       // time
#include "polygon.h"
using namespace std;


//----------    Hauptprogramm    ----------------------
int main()
{
    const Polygon a(19);
    Polygon b(a);
    const Point2D p1(0,0), p2(0.0f, 0.5f);
    cout << "Punktabstand " << dist(p1,p2) << endl; // Abstand zweier Punkte

    cout << "Umfang b: " << b.perimeter() << endl; // Umfang des geschlossenen Polygonzuges b
    b.append(p1);  //   Haenge den neuen Punkt als zusaetzliche, letzte Ecke an das Polygon b
    cout << "Umfang b: " << b.perimeter() << endl; // Umfang des geschlossenen Polygonzuges b

    cout << "Polynom mit " << a.number() << " Ecken" << endl; // Anzahl der Ecken des Polygons a
    cout << "Umfang a: " << a.perimeter() << endl; // Umfang des geschlossenen Polygonzuges a

// -----------------------------------------------------------------------------
//  Now, we demonstrate the difference in runtime between classes
//  Polygon_old (without mutable member) and
//  Polygon (with mutable member).
//  Be sure to switch on optimization (-O3  in my Release version).
//  -----------------------------------------------------------------------------
    vector<Polygon_old>  po;
    vector<Polygon>      pn;
    const int NN=100000;
    po.reserve(NN);                  // avoid multiple memory allocations in following loop
    pn.reserve(NN);
    /* initialize random seed: */
    srand (time(nullptr));
    for (int i=0; i<NN; ++i)
    {
        int kk = rand() % 1000 + 10;
        po.push_back(Polygon_old(kk));
        pn.push_back(Polygon(kk));
    }

    clock_t t;
    float   time_old, time_new;

    //  sort  the without mutable in class
    t = clock();
    sort(po.begin(),po.end());
    time_old = static_cast<float>(clock() - t)/CLOCKS_PER_SEC;


    //  sort  the with mutable in class
    t = clock();
    sort(pn.begin(),pn.end());
    time_new = static_cast<float>(clock() - t)/CLOCKS_PER_SEC;

    cout << "timings    no mutable: " << time_old  << " sec.;      mutable: " <<  time_new << " sec.\n";


    return 0;
}
```

```cpp
#include "polygon.h"

#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

ostream& operator<<(ostream& s, const Point2D& rhs)
{
    s << "(" << rhs.GetX() << "," << rhs.GetY() <<")";
    return s;
}

//float dist(const Point2D& a, const Point2D& b)
//{
//    return  sqrt( pow(a.GetX()-b.GetX(),2) + pow(a.GetY()-b.GetY(),2) );
//}

//------------------------------------------------------------------------------
Polygon_old::Polygon_old(int n)
    :  _v(n)
{
    for (unsigned int k=0; k<_v.size(); ++k)
    {
        _v.at(k) = Point2D( cos(k*2*M_PI/n), sin(k*2*M_PI/n) );
    }

//    copy(_v.begin(),_v.end(), ostream_iterator<Point2D>(cout," "));
}

float Polygon_old::perimeter() const
{
    float sum=dist( _v.front(),_v.back() );  // geschlossener Polygonzug
    for (unsigned int k=1; k<_v.size(); ++k)
    {
        sum += dist( _v[k], _v[k-1] );
    }
    return sum;
}

//------------------------------------------------------------------------------
```

Neuberechnung bei jedem Aufruf

```
Polygon::Polygon(int n)
    :  _v(n), _peri(-1.0f)
{
    for (unsigned int k=0; k<_v.size(); ++k)
    {
        _v.at(k) = Point2D( cos(k*2*M_PI/n), sin(k*2*M_PI/n) );
    }

//    copy(_v.begin(),_v.end(), ostream_iterator<Point2D>(cout," "));
}

float Polygon::perimeter() const
{
    if ( _peri<0.0f )
    {
        _peri=dist( _v.front(),_v.back() );  // geschlossener Polygonzug
        for (unsigned int k=1; k<_v.size(); ++k)
        {
            _peri += dist( _v[k], _v[k-1] );
        }
    }

    return _peri;
}
```

Berechne Umfang nur einmal, sonst nimm berechneten Wert

```cpp
#ifndef POLYGON_H_INCLUDED
#define POLYGON_H_INCLUDED

#include <iostream>
#include <vector>
#include <cmath>

/// @brief Class containing a point in 2D
///
class Point2D
{
  public:
      /// @brief Constructor without parameters.
      ///  Defines the point to the origin (0.0).
      ///
    Point2D() : _x(0.0f), _y(0.0f) {}

    /// @brief Constructor
    ///
    /// @param[in]  x coordinate in x direction
    /// @param[in]  y coordinate in y direction
    ///
    Point2D(float x, float y) : _x(x), _y(y) {}

    /// @brief Getter
    /// @return x coordinate
    float GetX() const {return _x;}

    /// @brief Getter
    /// @return y coordinate
    float GetY() const {return _y;}

private:
    float _x; //!< x coordinate of point
    float _y; //!< y coordinate of point
};

/// @brief Output operator for class @p Point2D
///
/// @param[in,out]  s    output stream
/// @param[in]      rhs  class instance
/// @return output stream
///
std::ostream& operator<<(std::ostream& s, const Point2D& rhs);

/// @brief Calculates the Euclidian distance between two points in 2D
///
/// @param[in] a  first point
/// @param[in] b  second point
/// @return Euclidian distance
///
// float dist(const Point2D& a, const Point2D& b);
inline float dist(const Point2D& a, const Point2D& b)
{
    return  std::sqrt( std::pow(a.GetX()-b.GetX(),2) + std::pow(a.GetY()-b.GetY(),2) );
}

//-------------------------------------------

class Polygon_old
{
  public:
    Polygon_old(int n);
    void append(const Point2D& a) { _v.push_back(a); }
    int number() const { return _v.size(); }
    float perimeter() const;
    bool operator<(const Polygon_old& rhs) const { return perimeter() < rhs.perimeter(); }

  private:
    std::vector<Point2D> _v;
};

//-------------------------------------------
/// \brief   Contains the description of a polygon, now with mutable.
///          The traverse is stored.
///
class Polygon
{
  public:
      /// @brief Constructs a regular polygon with vertices on the unit circle.
      ///
      /// @param[in]  n  number of vertices in the polygon
```

Umfangsberechnung

```
   /// @warning We use a mutable member @p _peri which should be defined -1 whenever the class instance has been changed.
   ///
  Polygon(int n);

/// @brief Adds a vertex to the end of the polygon traverse.
///
/// @param[in] a  2d point to add.
///
  void append(const Point2D& a) { _v.push_back(a); _peri=-1.0f; }

  /// @brief Number of vertices in polygon
  ///
  /// @return Number of vertices of the open polygon
  ///
  int number() const { return _v.size(); }

  /// @brief Computes the perimeter of the closed polygon
  ///
  /// @return Perimeter of the closed polygon
  /// @warning Uses a mutable variable
  ///
  float perimeter() const;

  /// @brief Less operator regarding the perimeter.
  ///
  /// @param[in] rhs  second polygon.
  /// @return True iff perimeter of recent instance is less than the perimeter of the second instance.
  ///
  bool operator<(const Polygon& rhs) const { return perimeter() < rhs.perimeter(); }

 private:
  std::vector<Point2D> _v;     //!< ordered vertices of the polygon
  mutable float    _peri;  //!< stores the perimeter once it is (re-)calculated
};




```

_peri wird in Methode verändert

```

#endif // POLYGON_H_INCLUDED
```