```cpp
#include <iostream>
#include <sstream>
#include <vector>
#include <lapacke.h>
using namespace std;

int main()
{
    int n = 10;
    unsigned int nodes = n+1;
    int rhs = 1;
    double a = 1.0;
    double alpha = 2.0;
    double gB = 1.0;

    double d = n + a/3.0/n;
    double s = - n + a / 6.0 / n;

    cout << d << endl;
    cout << s << endl;

    //Lapacke overwrites upper/lower diagonal so store twice
    vector<double> upperdiagonal(n,s);
    vector<double> lowerdiagonal(n,s);
    vector<double> diagonal(nodes,2*d);
    vector<double> F(nodes,1.0/n);

    diagonal[0]=1.0;
    diagonal[n]=d+alpha;

    upperdiagonal[0] = 0.0;
    //

    F[0] = 0;
    F[n]= 1.0/n/2.0 +alpha*gB;


    //Tridiagonal
    LAPACKE_dgtsv(LAPACK_COL_MAJOR,nodes,rhs,lowerdiagonal.data(),diagonal.data(),upperdiagonal.data(),F.data(),nodes);


    cout << "Solution u_h at nodes x_1..x_" << n << ":\n";
    for (unsigned int i = 0; i < nodes; ++i) {

        double xi = double(i) /double(n);
        cout << "u_h(" << xi << ") = " << F[i] << "\n";
        // F will be overwritten with sol
    }

    return 0;


}
```

```cpp

#include <iostream>
#include <sstream>
#include <vector>
#include <cmath>
#include <lapacke.h>
using namespace std;

int n = 10;
double threshold = 1.0/sqrt(2);
double lambda(double x)
{
    if(x < threshold)
    {
        return 1;
    }
    return 10;
}

double fillOffDiagonal(unsigned int i)
{

    double x_l = i/double(n);
    double x_u = (i+1)/double(n);

    if(x_l < threshold && x_u > threshold)
    {
        return -n*n*(threshold-x_l+10*(x_u-threshold));
    }
    return -n*lambda(x_l);
}

double fillDiagonal(int i)
{
    double x_l = (i-1)/double(n);
    double x_u = (i+1)/double(n);
    if(x_l < threshold && x_u > threshold)
    {
        return n*n*(threshold-x_l+10*(x_u-threshold));
    }
    return 2*n*lambda(x_l);
}

int main()
{
    //Ignored the first Row of K cause we now that u(0)=0

    int rhs = 1;
    unsigned int nodes = n+1;


    //Lapacke overwrites upper/lower diagonal so store twice
    vector<double> upperdiagonal(n,0.0);
    vector<double> lowerdiagonal(n,0.0);
    vector<double> diagonal(nodes,1.0);
    vector<double> F(nodes,0.0);

    for(int i=0; i < n; i++)
    {
        upperdiagonal[i] = fillOffDiagonal(i);
        lowerdiagonal[i] = fillOffDiagonal(i);
        diagonal[i+1] = fillDiagonal(i+1);
    }

    upperdiagonal[0]=0.0;
    lowerdiagonal[n-1]=0.0;
    diagonal[n] = 1.0;
    F[n] = 1.0;
```

keine globalen Variablen benutzen!

else

else

```
    69:
    70:
    71:     //Tridiagonal
    72:     LAPACKE_dgtsv(LAPACK_COL_MAJOR,nodes,rhs,lowerdiagonal.data(),diagonal.data(),up
perdiagonal.data(),F.data(),nodes);
    73:
    74:
    75:     cout << "Solution u_h at nodes x_0..x_" << n << ":\n";
    76:
    77:     for (unsigned int i = 0; i < nodes; ++i) {
    78:
    79:         double xi = double(i) /double(n);
    80:         cout << "u_h(" << xi << ") = " << F[i] << "\n";
    81:         // F will be overwritten with sol
    82:     }
    83:     return 0;
    84:
    85:
    86: }
```

```cpp

#include <iostream>
#include <sstream>
#include <vector>
#include <cmath>
#include <lapacke.h>
using namespace std;


int main()
{
    unsigned int ns[5] = {10,20,30,40,70};
    int p = 70;
    int rhs = 1;
    for(unsigned int i = 0; i< sizeof(ns)/sizeof(ns[0]); i++)
    {
        int n=ns[i];
        unsigned int nodes = n+1;

        vector<double> upperdiagonal(n,-n+p/2.0);
        vector<double> lowerdiagonal(n,-n-p/2.0);
        vector<double> diagonal(nodes,2*n);
        vector<double> F(nodes,0.0);
        upperdiagonal[0]=0.0;
        lowerdiagonal[n-1]=0.0;
        diagonal[0] = 1.0;
        diagonal[n] = 1.0;
        F[n] = 1.0;

        //Tridiagonal
        LAPACKE_dgtsv(LAPACK_COL_MAJOR,nodes,rhs,lowerdiagonal.data(),diagonal.data(
),upperdiagonal.data(),F.data(),nodes);


         cout << "Solution u_h at nodes x_0..x_" << n << ":\n";

        for (unsigned int j = 0; j < nodes; ++j) {

            double xi = double(j) /double(n);
            cout << "u_h(" << xi << ") = " << F[j] << "\n";
            // F will be overwritten with sol
        }
    }

    /*
    From Methode  der finiten Elemente fuer Ingenieure:
    small n approximates pu' not good espicially if n < p/2=35 therefor this oscilla
tion (see plot) occurs.
    */
    return 0;


}
```