```
#include <iostream>
#include "pladouble.h"
#include <cmath>

using namespace std;

//Initialisierung der PlaDouble-Internen Variablen (Belegung erfolgt spaeter)
int pladouble::switchnum;
double pladouble::tau = CXX_INF;
list <int> pladouble::active;
list <int> pladouble::activen;
list <double> pladouble::taulist;


//Funktion
pladouble fct(pladouble *x)
{   double gamma = 2;
    pladouble y = 0.0;
    y = fabs(1.0 - x[0])+ 10.0* fabs(x[1]-x[0]*x[0])+ sin(0.72*M_PI*(x[0]+x[1])) + gamma *
    fmax (fabs(x[0])+fabs(x[1])-10.0 , 0.0);

  return y;
}


//euklidische Norm
double norm(pladouble* x1, pladouble* x2,int Ddim){
    double y=0;
    for (int i=0; i<Ddim; i++){
    y=y+(x1[i].val-x2[i].val)*(x1[i].val-x2[i].val);

    }
    y= sqrt(y);
    return y;
}


//Funktion zur Auslagerung der Berechnung. x_start ist Startwert, d ist Multiplikator der
Richtung (-1 oder1)
pladouble* minsearch(pladouble *x_start, double d, int Ddim){
    pladouble* x_new = new pladouble[Ddim];
    pladouble* x_old = new pladouble[Ddim];
    x_new[0].val = x_start[0].val;
    x_new[1].val = x_start[1].val;
    x_new[0].grad = 3.0*d;
    x_new[1].grad = -1.0*d;

    double tau;
    pladouble y_old;
    pladouble y_new;
    double y_min;
    pladouble* x= new pladouble[Ddim];

    x_old[0].val=x_new[0].val;
    x_old[0].grad=x_new[0].grad;
    x_old[1].val=x_new[1].val;
    x_old[1].grad=x_new[1].grad;
```

```
    y_old=fct(x_old);
    y_min=y_old.val;
    int N = y_old.taulist.size();
//  cout<<"N = " << N << endl;
    double tau_min=0.0;

    for(int i=0; i<N; i++)
    {
      tau=y_old.taulist.front();
//    cout << "tau = " << tau << endl;
      y_old.taulist.pop_front();

      x[0].val=x_old[0].val+tau*x_old[0].grad;
      x[1].val=x_old[1].val+tau*x_old[1].grad;
      y_new= fct(x);
//    cout << "y_new = " << y_new.val << endl;
      if(y_new.val <= y_min)
      {
        y_min=y_new.val;
        tau_min=tau;
//      cout << "tau_min = " << tau_min <<endl;
      }
    }

        x_new[0].val= x_old[0].val+tau_min*x_old[0].grad;
        x_new[1].val= x_old[1].val+tau_min*x_old[1].grad;
    return x_new;


}



int main()
{
    //Dimension Bildbereich
    int Fdim = 2;
    //Dimension Definitionsbereich
    int Ddim = 2;
    //Epsilon
    double eps=0.0000001;

    pladouble* x_new = new pladouble[Ddim];
    pladouble* x_old = new pladouble[Ddim];
    //Startwert
    x_new[0].val = -2;
    x_new[1].val =  2;
    x_new[0].grad = 3.0;
    x_new[1].grad = -1.0;


    pladouble::switchnum = 0; //?

    //Iterationsschleife
    int steps = 0;
    do
```

```
    {
      steps = steps +1;
      x_old[0].val=x_new[0].val;
      x_old[0].grad=x_new[0].grad;
      x_old[1].val=x_new[1].val;
      x_old[1].grad=x_new[1].grad;

      pladouble* x_1=minsearch(x_new, 1.0, Ddim);
      pladouble* x_2=minsearch(x_new, -1.0, Ddim);

      pladouble y1 = fct(x_1);
      pladouble y2 = fct(x_2);
      if ( y1.val <= y2.val){
        x_new[0].val=x_1[0].val;
        x_new[0].grad=x_1[0].grad;
        x_new[1].val=x_1[1].val;
        x_new[1].grad=x_1[1].grad;
      }
      else{
        x_new[0].val=x_2[0].val;
        x_new[0].grad=x_2[0].grad;
        x_new[1].val=x_2[1].val;
        x_new[1].grad=x_2[1].grad;
      }

    }
    while(norm(x_new,x_old, Ddim)>eps);

    cout <<"NST bei: ("<<x_new[0].val << " , "<< x_new[1].val  <<")" << endl;
    cout <<"Dazu wurden " << steps << " Iterationsschritte benötigt" << endl;
}
```