```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h> // For the rand function
#include <time.h> // For the time function
#define min(x,y) ((x)<(y)?(x):(y))

double  UniformDist(double  a, double  b){ return a + (b-a)*(((double ) rand())/((double )  ↵
RAND_MAX)); }

/* f is the function to minimize. Minimum is at x=-0.19506755 */
#define c 0.2
#define s 14.5
#define fase 0.3
double  f(double  x){ return (x + c ) * x + cos( s * x - fase ) ; }
double  df(double  x){ return 2*x + c - s*sin( s * x - fase ); }
double  ddf(double  x){ return 2 - s*s*cos( s * x - fase ); }

// Defines the search region and function to control that we stay in it
#define bound 10
double  xbounded(double  x) { return ( (x > bound) ? bound : ( (x < -bound) ? -bound : x )  ↵
) ; }

/* Standard Next Configuration function
 * CRUCIAL: UniformDist(-dxmax,dxmax) (positive and negative) is important for  ↵
 transitivity in the state space.
 * We do not want to get out of bounds: dxmax = min(t*bound,bound) and xbounded.
 */
double  NextConfNoDeter(double  x, double  t){
  t=min(t*bound,bound); // Pass per value. t is not modifiede in main
  return xbounded(x + UniformDist(-t,t));
}

/* Here we dynamically take  into account deterministic information about the function
 * (weighted gradient -- Newton method applied to the derivative)
 * at the end of the process.
 * gradientborder defined what is "the end of the process" in terms of temperature.
*/
#define gradientborder 1.e-2
double  NextConfDynDeter(double  x, double  t){
  double dxmax=min(t*bound,bound);

  if(t >= gradientborder) return xbounded(x + UniformDist(-dxmax,dxmax));

  t = min(1.0,t/gradientborder); // Pass per value. t is not modifiede in main
  return x + t*UniformDist(-dxmax,dxmax) - (1.0-t)*df(x)/ddf(x);
}

/* Parameters of Simulated Annealing algorithm.
 * Notation follows François Bergeret and Philippe Besse document
*/
#define Lmax  1000 // This must allow to visit a good portion of the configuration space
#define Lamax 100
#define LTsw  0.0001
#define zerot 1.0e-12
#define BoltzmanKonstant    1.0 // Not important is equivalent to a change of units in t

// For the process of computing the maximum (melting) temperature
#define HTsw  0.95
#define tini  30.0
#define dt    1.0

// Scheduling program for the temperature: Exponential
#define alpha 0.95
double  exponentialsched(double  t, double  factor){ return t*factor; }

#define tol   1.e-14 // Precision of final result

float InnerLoop(double  *x, double  t){
        unsigned int L = 0, La = 0;
        double  xp, fx=f(*x), fxp;
        do{ L++; xp = NextConfDynDeter(*x, t); if(fabs(*x - xp) < tol) continue;
            fxp = f(xp);
            if(fxp <= fx || UniformDist(0.0, 1.0) < exp((fx-fxp)/(BoltzmanKonstant*t))) {  ↵
            *x = xp; fx = fxp; La++; }
        } while (L <= Lmax && La <= Lamax);
```

```
        return ((float) La)/((float) L);
} /* End of Inned Loop */

int main(){
        double  x, t=tini;
        float r;
    unsigned iter=0U;

        srand(time(0)); // Randomization
        x = UniformDist(-bound, bound); // Initial random seed

        while((r = InnerLoop(&x,t)) < HTsw) t += dt; // Loop to compute the melting temperature
        printf("\nMaximum temperature found: %lf (La/L = %f)\n\n", t, r);

// SA Algorithm --- Outer loop
    do { printf("Iteration %4u: temperature: %.8e (La/L = %9.6f); x = %20.16lf; f(x) = %20.16lf\n", iter, t, r, x, f(x));
             t = exponentialsched(t, alpha); iter++;
        }    while(t > zerot && (r = InnerLoop(&x,t)) > LTsw) ;
        if(t <= zerot) iter--;

       printf("\nBest minimum found: %20.14lf; f(.) = %20.16lf (t=%g; r=%f: iter=%u)\n\n",
       x, f(x), t/alpha, r, iter);
}
```