```cpp
#include <stdio.h>
#include <iostream>
#include <stddef.h>
#include <string>
#include <cstdlib>
#include <cstdarg>
#include <cmath>
#include <fstream>
#include <vector>
#include <cstdio>
#include <fstream>
#include <stdlib.h>
#include <sstream>

using std::cout;
using std::endl;
using std::ios;


// Function prototypes============================================================================================================
double Evaluate_dUdy_Plus(double lmix_plus, double y_pl, double R_pl);

std::vector <double> getUplus(double R_pl, int filewrite);

std::vector <double> evalFunc(double R_plus_est, int filewrite);

//==================================================================================================

// Main function
int main(){

  double U_ave=0.0;
  double tol = 1e-3; double toler = 1e-5;
  double Re_D = 40000;
  double R_plus_est = 0;
  double c = 0;
  double a= 1;
  double b = 10000;
  double root = 0;
  double ymax, Fmax, Udif;
  std::vector <double> f_a(4);
  std::vector <double> f_b(4);
  std::vector <double> f_c(4);
  std::vector <double> f_fin(4);
  double check_a, check_b, check_c;
  int iter=0;
  double Re_D_est;
  f_a = evalFunc(a, 0);
  f_b = evalFunc(b, 0);

  check_a = (2*f_a[3]*a - Re_D);
  check_b = (2*f_b[3]*b - Re_D);

  /* Check that that neither end-point is a root and if f(a) and f(b) have the same sign, throw an exception. */
  if ( check_a == 0 ){
    root = a;
    std::cout<<"root is "<<root<<endl;
  } else if ( check_b == 0 ){
    root = b;
    std::cout<<"root is "<<root<<endl;
  } else if ( check_a * check_b > 0 ){
    std::cout<<"f(a) and f(b) do not have opposite signs"<<endl;
  }

  // Begin the iterations-------------------------------------

  while ( fabs(Re_D_est - Re_D) > tol ){
```



```cpp
    //   for (int i = 0; i<10; i++){
    c = 0.5*(a + b);
    R_plus_est = c;

    f_a = evalFunc(a, 0);
    f_b = evalFunc(b, 0);
    f_c = evalFunc(c, 0);

    check_a = (2*f_a[3]*a - Re_D);
    check_b = (2*f_b[3]*b - Re_D);
    check_c = (2*f_c[3]*c - Re_D);
    U_ave = f_c[3];

    std::cout<<" a is = "<<a<<endl;
    std::cout<<"  b is = "<<b<<endl;
    std::cout<<"   c is = "<<c<<endl;
    std::cout<<" f(a) is = "<<check_a<<endl;
    std::cout<<"  f(b) is = "<<check_b<<endl;
    std::cout<<"   f(c) is = "<<check_c<<endl;
    std::cout<<"    -------------U_ave is: "<<U_ave<<endl;
    /* Check if we found a root or whether or not  we should continue with:
     *                 [a, c] if f(a) and f(c) have opposite signs, or
     *                 [c, b] if f(c) and f(b) have opposite signs. */

    if (check_c == 0){
      //          break;
    } else if (check_a * check_c < 0){
      b = c;
    } else{
      a = c;
    }

    if ( b - a < tol ){
      if ( fabs(check_a) < fabs(check_b) && fabs(check_a) < tol ){
        root = a;
        std::cout<<"root is "<<root<<endl;
      }
      else if ( fabs(check_b) < tol ){
        root = b;
        std::cout<<"root is "<<root<<endl;
      }
    }

    // Break the iterations if convergence tolerance has been met
    if (check_a < toler && check_b < toler && check_c < toler){
      break;
    }

    iter = iter + 1;

    Re_D_est = 2*U_ave*c;
  }
  std::cout<<" Final results Re_D = "<<2*U_ave*R_plus_est<<endl;
  std::cout<<" Final results R+ = "<<R_plus_est<<endl;
  std::cout<<"Number of iterations = "<<iter<<endl;
  f_fin = evalFunc(R_plus_est, 1);

  ymax = f_c[0];
  Fmax = f_c[1];
  Udif = f_c[2];
  U_ave = f_c[3];

  std::cout<<" ymax = "<<ymax<<" Fmax = "<<Fmax<<"  Udif = "<<Udif<<"  U_ave = "<<U_ave<<endl;
  return 0;
}

std::vector <double> evalFunc(double R_plus_est, int filewrite){

  std::vector <double> ResultsValue(4);
  //evaluate function!
```

```cpp
  ResultsValue = getUplus(R_plus_est, filewrite);

  double ymax = ResultsValue[0];
  double Fmax = ResultsValue[1];
  double Udif = ResultsValue[2];
  double U_ave = ResultsValue[3];

  return ResultsValue;

}

double Evaluate_dUdy_Plus(double lmix_plus, double y_pl, double R_pl){

  double dU_dy = 0.0;
  double lmix_plus_fourth = lmix_plus*lmix_plus*lmix_plus*lmix_plus;
  double y_over_R = (1 - y_pl/R_pl);

  if (y_pl <= 1){
    dU_dy = y_over_R - (y_over_R*y_over_R)*(lmix_plus*lmix_plus) +
2*(y_over_R*y_over_R*y_over_R)*(lmix_plus_fourth);
  }
  else if (y_pl > 1){
    dU_dy = ( std::sqrt((4*lmix_plus*lmix_plus*y_over_R) + 1)   - 1) /(2*lmix_plus*lmix_plus);
  }
  return dU_dy;
}

std::vector <double> getUplus(double R_pl, int filewrite){

  double dU_dy0, dU_dy1, dU_dy2, dU_dy3, dU_dy;
  std::vector <double> U_pl_profile;
  std::vector <double> Y_pl_profile;
  std::vector <double> ymax_profile;
  std::vector <double> Reynolds;
  std::vector <double> ReturnVals(4);

  double summ = 0.0;
  int counter = 0;
  double A0_pl = 26.0;
  double K = 0.41;
  double lmix_pl=0, lmix_pl1=0, lmix_pl2=0;
  double y_pl = 0;
  double delta_y_pl=0, U_ave = 0;
  double ymax=0, Fmax=0, Udif=0;


  double a = 2e-1, b = 2e-1;

  double U_n1 = 0, U_n2 = 0;

  while (y_pl <= R_pl){

    lmix_pl1 = K*y_pl*(1 - exp(-y_pl/A0_pl) );
    lmix_pl2 = 0.09*R_pl;

    if (lmix_pl1 <= lmix_pl2){
      lmix_pl = lmix_pl1;
      delta_y_pl = a;
    }
    else if (lmix_pl1 > lmix_pl2){
      lmix_pl = lmix_pl2;
      delta_y_pl = b;
    }

    /* =========  RK4 Scheme   ===================================================*/
    dU_dy0 = Evaluate_dUdy_Plus(lmix_pl, y_pl, R_pl);
    dU_dy1 = Evaluate_dUdy_Plus(lmix_pl + 0.5*delta_y_pl*dU_dy0, y_pl + 0.5*delta_y_pl, R_pl);
    dU_dy2 = Evaluate_dUdy_Plus(lmix_pl + 0.5*delta_y_pl*dU_dy1, y_pl + 0.5*delta_y_pl, R_pl);
    dU_dy3 = Evaluate_dUdy_Plus(lmix_pl + 0.5*delta_y_pl*dU_dy2, y_pl + delta_y_pl, R_pl);
```



```cpp
    if (y_pl > 0){
      U_n2 = U_n1 + (delta_y_pl/6)*(dU_dy0 + 2*dU_dy1 + 2*dU_dy2 + dU_dy3);
    }

    // update vectors to store these values
    U_pl_profile.push_back(U_n2);
    Y_pl_profile.push_back(y_pl);
    ymax_profile.push_back(dU_dy0 * lmix_pl);

    dU_dy = Evaluate_dUdy_Plus(lmix_pl, y_pl, R_pl);
    Reynolds.push_back(lmix_pl*lmix_pl * dU_dy*dU_dy);
    summ = summ + U_n2 * (1 - y_pl/R_pl ) * delta_y_pl;
    y_pl = y_pl + delta_y_pl;
    if (fabs(y_pl - R_pl) < 1){
      break;
    }

    counter = counter + 1;

    U_n1 = U_n2;
  }
  //===================================== known and complete
============================================
  U_ave = (2/R_pl) * summ;

  double val_max = 0, Rmax = 0;
  // now evaluate the new max values: these can only be evaluated once we know the entire velocity
profile, hence why we initialize at start
  for (int i=0; i< counter; i++){
    double ypl = Y_pl_profile[i];
    if (ymax_profile[i] > val_max){
        val_max = ymax_profile[i];
        if (ypl <= 1){
          delta_y_pl = a;
        }
        if (ypl > 1){
          delta_y_pl = b;
        }
        ymax = i * delta_y_pl;
        Fmax = (1/K)*val_max;
    }
    if (Reynolds[i] > Rmax){
        Rmax = Reynolds[i];
    }
  }

  Udif = U_pl_profile[counter-1]; // - U_pl_profile[counter * ymax/R_pl];

  ReturnVals[0] = ymax;
  ReturnVals[1] = Fmax;
  ReturnVals[2] = Udif;
  ReturnVals[3] = U_ave;

  // //============== now write results to file
(Tecplot!)================================================

  double num_elem = Reynolds.size();

  if (filewrite == 1){
    std::stringstream stream1, stream3, stream4, stream5, stream6, stream7;
    stream1 << "U_plus_two-layer_mixing_model.dat";
    stream3 <<"i="<<num_elem + 1;
    stream4<<"title = "<<"'"<<stream1.str()<<"'";
    std::string var1 = stream3.str();
    std::string var2 = stream4.str();
    std::string fileName1 = stream1.str();

    FILE* fout = fopen(fileName1.c_str(), "w");

    fprintf(fout, "%s", var2.c_str() ); fprintf(fout, "\n");
```

```
    fprintf(fout, "%s", "variables = 'U+/U+max' 'y+/R+' 'y+' 'U+'  'normalized_Reynolds_Shear_Stress' ");
    fprintf(fout, "\n"); //'Reynolds Shear Stress'
    fprintf(fout, "%s %s %s", "zone",var1.c_str(),"f=point"); fprintf(fout, "\n");  //

    summ = 0;
    for (int j = 0; j <= num_elem; j++){
      fprintf(fout, "%e\t %e\t %e\t %e\t %e\t", U_pl_profile[j]/U_pl_profile[num_elem], Y_pl_profile[j]/
      Y_pl_profile[num_elem],  Y_pl_profile[j], U_pl_profile[j],  Reynolds[j]/Rmax);
      fprintf(fout, "\n");
    }
    fclose(fout);
    std::cout<<" "<<endl; std::cout<<"U_plus results file successfully written"<<endl;

    //==================================================================================

    stream5 << "Laufer Experimental Data.dat";
    stream6 <<"i="<<11;
    stream7<<"title = "<<"'"<<stream5.str()<<"'";
    var1 = stream6.str();
    var2 = stream7.str();
    std::string fileName2 = stream5.str();
    fout = fopen(fileName2.c_str(), "w");
    fprintf(fout, "%s", var2.c_str() ); fprintf(fout, "\n");
    fprintf(fout, "%s", "variables = 'U+/U+max' 'y+/R+'  "); fprintf(fout, "\n");
    fprintf(fout, "%s %s %s", "zone",var1.c_str(),"f=point"); fprintf(fout, "\n");  //
    fprintf(fout, "%e\t %e\t ", 0.333, 0.010);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.696, 0.095);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.789, 0.210);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.833, 0.280);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.868, 0.390);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.902, 0.490);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.931, 0.590);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.961, 0.690);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.975, 0.800);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 0.999, 0.900);  fprintf(fout, "\n");
    fprintf(fout, "%e\t %e\t ", 1.000, 1.000);  fprintf(fout, "\n");
    fclose(fout);
    std::cout<<"Laufer results successfully written"<<endl;
  }

  return ReturnVals;

}
```