```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>
#include <algorithm>
#include <utility>

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


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

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

std::vector <double> evalFunc(double ymax, double Fmax, double Udif, double R_pl, int filewrite);

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

// Main function
int main(){

  // variables
  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 Fmax_new=0,             ymax_new=0,             Udif_new = 0,     U_ave_new =0;
  double Fmax_prev=0,            ymax_prev=0,            Udif_prev=0,      U_ave_prev=0;
 // From Question 2
  double ymax = 59.6, Fmax = 2.31695,  Udif = 23.0412,  U_ave = 18.2401;
  double delt_init =30;
  int coun = 0, iter=0;
  double Re_D = 40000, Re_D_est = 0;
  double c = 0;
  double a= 1;
  double b = 10000;
  double root = 0;

  double check_a = 0;
  double check_b = 0;
  double check_c = 0;

  double damp = 0.3;
  double tol = 1e-5; double toler = 1e-5;

  /*=================== BISECTION METHOD ==================================================== */


  f_a = evalFunc(ymax, Fmax, Udif, a, 0);
  f_b = evalFunc(ymax, Fmax, Udif, 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-------------------------------------

  double delt1;

  delt1 = delt_init;

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

//     std::cout<<"Outer iteration number: "<<iter<<endl;
    c = 0.5*(a + b);
    f_a = evalFunc(ymax, Fmax, Udif, a, 0);
    f_b = evalFunc(ymax, Fmax, Udif, b, 0);

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

    //===========================================================================
    // iterate for a converged ymax, Fmax, Udif
    //====================== BALDWIN LOMAX MODEL ===================================

    while ( delt1 > tol){
//        std::cout<<"Inner iteration number: "<<coun<<endl;
      // Update from previous timestep
      Fmax_prev = Fmax;
      ymax_prev = ymax;
      Udif_prev = Udif;
      U_ave_prev = U_ave;
      //==============================================================
      // new values for new n plus update for U+
      f_c = evalFunc(ymax_prev, Fmax_prev, Udif_prev, c, 0);

      check_c =  (2*f_c[3]*c - Re_D);

      // Update new variable values
      ymax_new = f_c[0];
      Fmax_new = f_c[1];
      Udif_new = f_c[2];
      U_ave_new = f_c[3];
      //==============================================================
      // Applying a damping factor before re-plugging in the values
      Fmax = 0; ymax = 0; Udif = 0; U_ave = 0;
      Fmax = Fmax_prev + damp*(Fmax_new - Fmax_prev);
      ymax = ymax_prev + damp*(ymax_new - ymax_prev);
      Udif = Udif_new;
      U_ave = U_ave_new;

      double delt1 = fabs(ymax - ymax_prev);
      double delt2 = fabs(Fmax - Fmax_prev);
```

```
      if (delt1 <=tol && delt2 <=tol ){
        break;
      }
      coun = coun + 1;
    }
    std::cout<<"    ymax = "<<ymax<<"   Fmax = "<<Fmax<<endl;
    std::cout<<" Number of ymax iterations is =    "<< coun<<endl;
    //=============================================================================

    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*c<<endl;
  std::cout<<" Final results R+ = "<<c<<endl;
  std::cout<<" Final U_ave = "<<U_ave<<endl;
  std::cout<<"Number of bisection iterations = "<<iter<<endl;
  f_fin = evalFunc(ymax, Fmax, Udif, c, 1);

  return 0;

}

std::vector<double> evalFunc(double ymax, double Fmax, double Udif, double R_pl, int filewrite){

  std::vector<double> ReturnVals;
  //evaluate function!
  ReturnVals = getUplus(ymax, Fmax, Udif, R_pl, filewrite);

  return ReturnVals;
}

double Evaluate_dUdy_Plus(double lmix_plus, double y_pl, double R_pl, double ymax, double Fmax, double Udif){

  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);
  double alpha = 0.0168;
  double Ccp = 1.6;
  double FKleb, Fwake, CKleb = 0.3, Cwk = 1;
  double visc=0, eddyIN, eddyOUT;
```

```
  if (y_pl <= 1){  // as it was!
    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{
    eddyIN = lmix_plus*lmix_plus * ( std::sqrt(4*lmix_plus*lmix_plus*y_over_R + 1)   - 1)
    /(2*lmix_plus*lmix_plus);

    double kleb_6 = (y_pl* CKleb/ymax) * (y_pl* CKleb/ymax) * (y_pl* CKleb/ymax) * (y_pl* CKleb/ymax) *
    (y_pl* CKleb/ymax) * (y_pl* CKleb/ymax);
    FKleb = 1/( 1 + 5.5 * kleb_6);

    if ( ymax*Fmax <= Cwk*ymax*Udif*Udif/Fmax){
      Fwake = ymax*Fmax;
    }
    else{
      Fwake = Cwk*ymax*Udif*Udif/Fmax;
    }
    eddyOUT = alpha * Ccp * Fwake * FKleb;

    if (eddyIN <= eddyOUT){    // maybe compare directly
      visc = eddyIN;   // inner layer
      dU_dy = y_over_R / (1 + visc);
    }
    if (eddyIN > eddyOUT){
      visc = eddyOUT;   // outer layer
      dU_dy = y_over_R / (1 + visc);
    }
  }

  return dU_dy;
}


std::vector <double> getUplus(double ymax, double Fmax, double Udif, 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> Reynolds;
  std::vector <double> ymax_profile;
  std::vector <double> Results(4);
  double summ = 0.0;     int counter = 0;
  double A0_pl = 26.0;   double K = 0.4;
  double y_pl = 0;       double lmix_pl, delta_y_pl;
  double U_n1=0, U_n2=0;
  double U_ave=0;

  double alpha = 0.0168;
  double Ccp = 1.6;
  double FKleb, Fwake, CKleb = 0.3, Cwk = 1;
  double visc=0, eddyIN, eddyOUT;

  double aa = 2e-1;
  double bb= 1e-1;

//   std::cout<<"Stuck here.... "<<endl;

  // Build the Velocity
  Profile===============================================================================
  while (y_pl <= R_pl){
```

```
    if (y_pl <= 2){
      delta_y_pl = aa;
    }
    if (y_pl > 2){
      delta_y_pl = bb;
    }


    lmix_pl = K*y_pl*(1 - exp(-y_pl/A0_pl) );
    double lmix_plus_fourth = lmix_pl*lmix_pl*lmix_pl*lmix_pl;

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

    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);

    double y_over_R = (1 - y_pl/R_pl);

    // a build for the Reynold's Stress
    if (y_pl <= 1){
      dU_dy = y_over_R - (y_over_R*y_over_R)*(lmix_pl*lmix_pl) +
2*(y_over_R*y_over_R*y_over_R)*(lmix_plus_fourth);
      Reynolds.push_back(lmix_pl*lmix_pl * dU_dy*dU_dy);
    }
    if (y_pl > 1){
      eddyIN = lmix_pl*lmix_pl * ( std::sqrt(4*lmix_pl*lmix_pl*y_over_R + 1)   - 1) /(2*lmix_pl*lmix_pl);
      double kleb_6 = pow((y_pl* CKleb/ymax),6);
      FKleb = 1/( 1 + 5.5 * kleb_6);

      if ( ymax*Fmax <= Cwk*ymax*Udif*Udif/Fmax){
        Fwake = ymax*Fmax;
      }
      else{
        Fwake = Cwk*ymax*Udif*Udif/Fmax;
      }
      eddyOUT = alpha * Ccp * Fwake * FKleb;

      if (eddyIN <= eddyOUT){
        visc = eddyIN;   // inner layer
        dU_dy = y_over_R / (1 + visc);
      }
      if (eddyIN > eddyOUT){
        visc = eddyOUT;   // outer layer
        dU_dy = y_over_R / (1 + visc);
      }
      Reynolds.push_back(visc * dU_dy);
    }

    summ = summ + U_n2 * (1 - y_pl/R_pl ) * delta_y_pl;
    counter = counter + 1;
    y_pl = y_pl + delta_y_pl;
    if (fabs(y_pl - R_pl) < 1){
      break;
```

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

  double val_max = 0, Rmax = 0, val_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 <= 2){
          delta_y_pl = aa;
        }
        if (ypl > 2){
          delta_y_pl = bb;
        }
        ymax = i * delta_y_pl;
        Fmax = (1/K)*val_max;
      }
      if (Reynolds[i] > Rmax){
        Rmax = Reynolds[i];
      }
  }
  double num_elem = U_pl_profile.size();
  Udif = U_pl_profile[num_elem]; // - U_pl_profile[counter * ymax/R_pl];

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

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

  if (filewrite == 1){
    //     std::cout<<"U_max is "<<U_pl_profile[counter-1]<<endl;

    std::stringstream stream1, stream3, stream4, stream5, stream6, stream7;
    stream1 << "U_plus_two-layer_mixing_model.dat";
    stream3 <<"i="<<num_elem;
    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 Results;

}
```