Code of C++ with ILOG CPLEX version 12.8

```
#include <time.h>
#include <ctime>
#include <sstream>
#include <math.h>
#include <stdio.h>
#include <ilcplex/ilocplex.h>
#include <ctime>


//define global constants
#define infinity 10000000000
#define MODLUS 2147483647
#define MULT1      24112
#define MULT2      26143


ILOSTLBEGIN
IloEnv env;

bool deterministic_equivalent;

int N = 324;  // Number of nodes
int P = 2; //Number of facilities to locate
int S = 1200; //coverage radius

IloRangeArray scenarioconstsets(env);

IloExprArray expression(env);
IloExpr expr(env);
IloExpr expr2(env);
IloExpr expr1(env);
IloExpr deterministic_cost(env);

IloNumArray prob(env, 1);

ofstream output("facility_output.txt");


string getvarname_x(int j)
{
       char ch[40];

       sprintf_s(ch, "x%d", j);
       return(ch);
}
string getvarname_z(int i)
{
       char ch[40];

       sprintf_s(ch, "z%d", i);
       return(ch);
}
int index(int i, int j)
{
       return(i * N + j);
```

```
}
void const1(IloModel model, IloIntVarArray x)
{
	int j;
	for (j = 0; j < N; j++)
	{
		expr += x[j];
	}

		scenarioconstsets.add(expr == P);
		expr.clear();

}

void const2(IloModel model, IloIntVarArray z, IloIntVarArray x, const IloNumArray d)
{

	int i, j;
	//const IloNumArray d;
	//cout << "7667" << endl;
	//cout << d << endl;

	for (j = 0; j < N; j++)
	{
		//cout << "7667" << endl;
		expr1 = expr1 + z[j];
		for (i = 0; i < N; i++)
		{

			//gcout << expr2 << endl;
			//cout << d[index(i, j)] << endl;
			if (d[index(j, i)] <= S)
			{

				expr1 += -x[i];
				//cout << expr2 << endl;

			}

		}scenarioconstsets.add(expr1 <= 0);
		expr1.clear();

	}

}

void scenario_cost(IloModel model,const IloNumArray h, IloIntVarArray z)
{

	for (int i = 0; i < N; i++)
	{
		expr +=h[i] * z[i];
		//cout << "go" << endl;
	}

	deterministic_cost +=  expr;

}
```

```
void solution_values(IloCplex cplex, IloIntVarArray x, IloIntArray x_var, IloIntVarArray
z, IloIntArray z_var)
{
	int sirma;
	env.out() << "Solution status = " << cplex.getStatus() << endl;
	env.out() << "Solution value = " << cplex.getObjValue() << endl;

	//dem_objval = cplex.getObjValue();
	//cout << dem_objval;
	int i,j;
		for (i = 0; i < N; i++)
		{
			z_var[(i)] = cplex.getValue(z[(i)]);
			output << "z" <<"," << i << "=" << " " << z_var[(i)] << endl;

		}
		for (j = 0; j < N; j++)
		{
			x_var[(j)] = cplex.getValue(x[(j)]);
			output << "x" << "," << j << "=" << " " << x_var[(j)] << endl;

		}

}

void model(const IloNumArray h,const IloNumArray d)
{
	int sirma;

	int i,j;
	string var_name;
	IloTimer elapsed_time(env);
	//IloNum dem_objval; //objective value for DEM

	IloIntArray x_var(env, N);
	IloIntArray z_var(env, N);


	if (deterministic_equivalent)
	{

		elapsed_time.start();
		IloIntVarArray x(env, N, 0, 1);
		IloIntVarArray z(env, N, 0, 1);

		// give names to the variables

		for (i = 0; i < N; i++)
		{
			var_name = getvarname_z(i);
			z[i].setName(var_name.c_str());
		}
		for (j = 0; j < N; j++)
		{

			var_name = getvarname_x(j);
			x[j].setName(var_name.c_str());
```

```
		}


		IloModel model(env);

		//firstapp(model, x);

			//constraints
			const1(model,x);
			const2(model,z,x,d);

			//calculate scenario cost
			scenario_cost(model,h,z);
			expr.clear();


		model.add(scenarioconstsets);
		IloObjective objective = IloMaximize(env, deterministic_cost);
		model.add(objective);

		IloCplex cplex(model);
		//cplex.setOut(env.getNullStream()); // turn off the cplex screen outputs
		cplex.exportModel("facilitymodel.lp");

		//	cplex.setParam(cplex.EpGap, 0.0008); // for testing purposes
		//  cplex.setParam(cplex.TiLim, 3600);
		cplex.solve();

				cout << "elapsed time: "<< elapsed_time.getTime() << endl;
		elapsed_time.reset();

		solution_values(cplex, x, x_var, z, z_var);
		scenarioconstsets.end();
		deterministic_cost.end();
		deterministic_cost.end();
		cplex.end();
		objective.end();
		model.end();
		//cout << 6666 << endl;
		cin >> sirma;
	}
}

int main(int, char**)
{
	int i, j;
	int start_s = clock();
		deterministic_equivalent = 1; //deterministic equivalent model is being
solved

	////////////////////////////////////////////////////////////////////////////
	// now define all parameters necessary for the model
	IloNumArray d(env, 1); // distance btw i and j
	IloNumArray h(env, 1);  // demand amount

//	cin >> sirma;

	d.setSize(N*N);
```

```
	h.setSize(N);
		//////////////////////////////////////////////////////
	///ADD INPUT DATA!!!! ///////

int	ind = 0;
int	ind2 = 0;
	ifstream in;
	ifstream in2;
	in.open("SJC324_demand.txt");
		for (ind = 0; ind < N; ind++)
		{
			in >> h[ind];
			cout << h[ind]<<endl;
		}
		in.close();

	//induction durations
		in2.open("SJC324_distance.txt");

		for (j = 0; j < N; j++)
		{
			for (i = 0; i < N; i++)
			{

				//int i = 0;

				in2 >> d[index(j, i)];

				//cout << e[ind2] << endl;
			}
		}
		in2.close();
	model(h,d);
	env.end();
	return 0;
	int stop_s = clock();
	double time = (stop_s - start_s) / double(CLOCKS_PER_SEC) * 1000;
	cout << "total time:"<<time << endl;
}
```