ภาคผนวก ก
ตัวอย่างตารางเมตริกซ์ระยะทางระหว่างตำแหน่งเก็บรวบรวมขยะต่างๆกับศูนย์11ไร่

ภาคผนวก ข
โปรแกรมภาษาC++ ที่ใช้คำนวณจัดเส้นทางการเก็บรวบรวมขยะ ของแนวทางที่ 1 และแนวทางที่ 2

โปรแกรม C++ แนวทางที่ 1 วิธีการแก้ปัญหาที่ดีที่สุดสำหรับปัญหาTSP

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iomanip>
#include <fstream>
//#include <sys/time.h>
#include <ilcplex/ilocplex.h>
ILOSTLBEGIN //ILOG standard template library
typedef IloArray<IloNumVarArray> IloNumVarArray2;  //define a 2D variable array.
typedef IloArray<IloNumArray> IloNumArray2; // define a 2D array.
using namespace std;
void insertion_project(int*& routeinst,float** distance,float max,int numbercity);
float totaldistance(float** distance,int* route,int numbercity);
void twoopt_project(int**&edge,float** distance,int*& routeinst,int numbercity);
void TSPsolve(float**distance,int*nod,int*checknod,int countsupplier,
      float&mintourlength,int numbersupplier);
int main()
{clock_t start,end;
  unsigned seed;// = time(NULL);//must block "=time(NULL)"for random differently
 float mintourlength,max=10000.0;
 int numbercity,i,j,k;
 int*routeinst,*checknod;;
 int**edge;
 float**distance;
 double diff=0.0;
 ifstream readfile;
 ofstream outputfile;
 cout<<"Enter seed = ";  //**random differently
 cin>>seed;             //**random differently
 srand(seed);           //**end random differently
 cout<<endl<<"Total number of cities including a plant: ";
 cin>>numbercity;
 distance = new float*[numbercity];
 edge = new int*[numbercity];
 checknod=new int[numbercity];
 for(i=0; i<numbercity; i++){
     distance[i] = new float[numbercity];
     edge[i]= new int[numbercity];
 }
 routeinst = new int[numbercity];
 cout.precision(5);
 cout<<endl;
readfile.open("waste_358_collecting_points.out");  // 358+1=359 nodes including depot
outputfile.open("result_waste_6.out");
for( i=0; i<numbercity; i++)
 {  for( j=0; j<numbercity; j++)
   {   readfile>>distance[i][j];
       cout<<distance[i][j]<<" ";
   }
   cout<<endl;
   routeinst[i]= i;
 }
 insertion_project(routeinst,distance,max,numbercity);
 cout<<"TSP TOUR of vehicle from insertion:";
 for(j=0;j<numbercity;j++)
 {  cout<<routeinst[j]<<"-";
 }
 cout<<routeinst[0]<<endl;
 cout<<"The total distance = "<<totaldistance(distance,routeinst,numbercity);
 cout<<endl;
 cin>>i;
 twoopt_project(edge,distance,routeinst,numbercity);
 cout<<"TSP TOUR of vehicle from two-opt: ";
 for(j=0;j<numbercity;j++)
 {  cout<<routeinst[j]<<"-";
 }
 cout<<routeinst[0]<<endl;
 cout<<"The total distance = "<<totaldistance(distance,routeinst,numbercity);
 cout<<endl;
 cin>>i;
 start=clock(); //count time to solve for TSP optimally
 TSPsolve(distance,routeinst,checknod,(numbercity-1),mintourlength,numbercity);
 end=clock();
 diff=(end-start)/CLOCKS_PER_SEC;
 cout<<"The total distance = "<<mintourlength;
 cout<<endl;
 //cout<<"diff="<<diff<<"  end="<<end<<"  start="<<start<<"
CLOCKS_PER_SEC="<<CLOCKS_PER_SEC<<endl;
 cout<<"Average Run time of Optimal TSP = "<<diff<<endl;
 cin>>i;
 readfile.close();
 outputfile.close();
 for(i=0;i<numbercity;i++)
 {   delete[]distance[i];
     delete[]edge[i];
 }
 delete[]distance;
 delete[]edge;
 delete[]routeinst;
 delete[] checknod;
     return 0;
}
//*************************************************************************
//**********************TOTALDISTANCE FUNCTION****************************
//totaldistance function calculates the total distance of the tour.
//*************************************************************************
//i = a counter to control looping.
//TTD = the total distance of the tour.
//*************************************************************************
float totaldistance(float** distance,int* route, int numbernode)
{  int i;
   float TTD = 0.0;
   for(i=1; i<numbernode; i++)   //Calculate the total travel distance of a tour.
   {     TTD = TTD + distance[route[i-1]][route[i]];
   }
   TTD = TTD+distance[route[--i]][route[0]];
   return TTD;
}
//****************************************************************************
//****************************************************************************
//**************************Insertion Function ******************************
//insertion function generates a tour using the arbitrary insertion heuristic.
//****************************************************************************
//count,count1,count2,i,k,ii = counters to control looping.
//decre = decreasing cost(distance) due to node insertion.
//hold = a variable to hold a value of routeinst[i].
//incre = increasing cost(distance) due to node insertion.
```

```
//insert1, insert2 = indexes of nodes.
//insertcost[i][j] = an array of total cost increased due to node insertion between nodes i and j.
//node[i] = a node with an index i, temporarily used.
//numbernode = the total number of nodes including a warehouse.
//r =  a random number.
//routeinst[i]= an array of the sequence of nodes in a route generated by the
//            arbitrary insertion heuristic.
//select = a node selected to insert between two other nodes in a subtour.
//***************************************************************************
void insertion_project(int*& routeinst,float** distance,float max,int numbercity)
{ float**insertcost, mininst = sqrt(2)*max, incre, decre, mincost;
  int count1, insert1, insert2, select, tempnode, firstnode,
     count=2,i,k,ii,r,hold,numbernode;
  numbernode=numbercity;
// Form a subtour consisting a warehouse and the nearest node.
  insertcost = new float*[numbercity];
  for(i=0;i<numbercity;i++)
    insertcost[i]= new float[numbercity];
  for( i=1; i<numbernode; i++)    //*****start block if not random
  {  if(distance[routeinst[0]][routeinst[i]] <= mininst)
    {  mininst = distance[routeinst[0]][routeinst[i]];
       firstnode = i;
    }
  }
  tempnode = routeinst[firstnode];
  routeinst[firstnode]= routeinst[1];
  routeinst[1]= tempnode;
  while( count < numbernode)
  {   r = rand()%(numbernode-count)+count;
     hold = routeinst[r];
     routeinst[r] = routeinst[count];
     routeinst[count] = hold;
     cout<<routeinst[count]<<"-";
     count++;
  }             //end block ************************************
  cout<<endl;
//Arbitratrily select the node to insert in the subtour.
  for(count1 = 1; count1 < (numbernode-1); count1++)
  {  select = routeinst[count1+1];
//Calculate the insertion cost of the edge comprising a warehouse and a node last visited.
    incre = distance[routeinst[count1]][select]+distance[select][routeinst[0]];
    decre = distance[routeinst[count1]][routeinst[0]];
    insertcost[routeinst[count1]][routeinst[0]]= incre - decre;
    mincost = insertcost[routeinst[count1]][routeinst[0]];
    insert1 = count1;
    insert2 = 0;
//Calculate the insertion cost for each remaining edge.
    for( k=0; k<count1; k++)
    {  incre = distance[routeinst[k]][select]+distance[select][routeinst[k+1]];
       decre = distance[routeinst[k]][routeinst[k+1]];
       insertcost[routeinst[k]][routeinst[k+1]]= incre - decre;
       if(insertcost[routeinst[k]][routeinst[k+1]] <= mincost)
       {   mincost = insertcost[routeinst[k]][routeinst[k+1]];
          insert1 = k;
          insert2 = k+1;
       }
    }
//Update the order of nodes visited in the subtour after insertion.
    if(insert1 != count1)
    {  for( ii=count1; ii >= insert2; ii--)
          routeinst[ii+1] = routeinst[ii];
    }
    routeinst[insert1+1] = select;
  }
  for(i=0;i<numbercity;i++) delete[]insertcost[i];
  delete[]insertcost;
}
//**************************************************************************
//**************************************************************************
//twootp function improve the solution obtained from the arbitrary insertion.
//**************************************************************************
//count,i,ii,iii,j,jj,k = counters control looping.
//decreasecost = cost decreased due to disconnection of pairs of nodes.
//edge[i][j] = 1 if node i is connected with node j or 0, otherwise.
//increasecost = cost increased due to connection of two pairs of nodes.
//maxsave = highest cost reduction obtained from two-edge exchange.
//save = total cost reduction due to exchange of two edges.
//selectnode = index of the node to be connected in two-edge exchange.
//temproute[i] = a temporary array to hold values of an array routeinst.
//**************************************************************************
void twoopt_project(int**&edge,float** distance,int*& routeinst,int numbercity)
{int*temproute;
 int count,i,ii,iii,j,jj,k,m,selectnode,numbernode;
 float decreasecost,increasecost,maxsave,save;
 numbernode=numbercity;
 temproute = new int[numbernode+2];
 maxsave = 1.0;
 i = 1;
 for(ii=0;ii<numbernode;ii++)
 {   for(jj=0;jj<numbernode;jj++)
       edge[routeinst[ii]][routeinst[jj]]= 0;
 }
//Initialize edge[i][j] equal to 1 if nodes i and j are connected.
 for(iii=0;iii<numbernode-1;iii++)
 {   edge[routeinst[iii]][routeinst[iii+1]]= 1;
    edge[routeinst[iii+1]][routeinst[iii]]= 1;
 }
 edge[routeinst[numbernode-1]][routeinst[0]]= 1;
 edge[routeinst[0]][routeinst[numbernode-1]]= 1;
 while( i <= numbernode )
 {if(maxsave > 0)
  {   for(jj=1;jj<=numbernode;jj++)
      {   temproute[jj]= routeinst[jj-1];
      }
      temproute[numbernode+1]= routeinst[0];
      temproute[0]= routeinst[numbernode-1];
  }
  maxsave = 0.0;
  for(j=1;j<=numbernode;j++)
  {   if((edge[temproute[i+1]][temproute[j]]==0)&&(temproute[i+1]!=temproute[j]))
      { decreasecost = distance[temproute[i]][temproute[i+1]]+distance[temproute[j]][temproute[j-1]];
        increasecost = distance[temproute[i+1]][temproute[j]]+distance[temproute[i]][temproute[j-1]];
        save = decreasecost-increasecost;
        if(save>maxsave)
        {   maxsave = save;
            selectnode = j;
        }
      }
  }
  if(maxsave>0.0)       // Exchange two edges if it results cost reduction.
  {  edge[temproute[i]][temproute[i+1]]= 0;
     edge[temproute[i+1]][temproute[i]]= 0;
    edge[temproute[selectnode-1]][temproute[selectnode]]= 0;
    edge[temproute[selectnode]][temproute[selectnode-1]]= 0;
    edge[temproute[i+1]][temproute[selectnode]]= 1;
    edge[temproute[selectnode]][temproute[i+1]]= 1;
    edge[temproute[i]][temproute[selectnode-1]]= 1;
    edge[temproute[selectnode-1]][temproute[i]]= 1;
    count = 1;
    temproute[0]= routeinst[0];
    while(count < numbernode) // Find route connection for each pair of nodes.
    {   for(k=1;k<numbernode;k++)
       {   if(edge[temproute[count-1]][routeinst[k]]== 1)
          {  temproute[count]= routeinst[k];
             edge[routeinst[k]][temproute[count-1]]= 0;
             count++;
             break;
           }
       }
     }
     for(ii=1;ii<numbernode;ii++)   //Print a new route.
     {   routeinst[ii]= temproute[ii];
        edge[routeinst[ii]][routeinst[ii-1]]= 1;
     }
  }
  if( maxsave > 0 )
    i = 1;     //Start a next iteration.
  else
    i++;       //Continue searching for edge exchage in the current iteration.
 }
 delete[] temproute;
}
//****************************************************************************
//**************************************************************************
//*************************TSPsolve function*****************************
//Use CPLEX to solve TSP optimally by formulating TSP as IP without subtour
//elimination constraints and then check the solution from CPLEX if there
//is any subtour. If so, add a constraint to eliminate that subtour.
//**************************************************************************
//**************************************************************************
void TSPsolve(float**distance,int*nod,int*checknod,int countsupplier,
      float&mintourlength,int numbersupplier)
{  int countnod,endnod,firstnod,numbersubtour,subtour,totalnod,i,j,k,m,n,numberconstraintadd=0;;
   float**distancecoef;
   int*subroute,*insubtour;
   subroute=new int[countsupplier+2];
   insubtour=new int[countsupplier+1];
   distancecoef = new float*[(countsupplier+1)];
   for(i=0;i<(countsupplier+1);i++){
    distancecoef[i]= new float[(countsupplier+1)];
   }
   IloEnv env;
   IloModel model(env);
   IloCplex cplex(env);
   IloObjective objective=IloAdd(model,IloMinimize(env));
   IloRangeArray degreeconstraint=IloAdd(model,IloRangeArray(env,(countsupplier+1),2,2));
   IloNumVarArray2 variablex(env,(countsupplier+1));//create 2 dimensional array.
   for(i=0;i<(countsupplier+1);i++)
   {  variablex[i]=IloNumVarArray(env,(countsupplier+1),0,1,ILOBOOL);
   }
   for(i=0;i<(countsupplier+1);i++)
   {  for(j=0;j<(countsupplier+1);j++)
      { distancecoef[i][j]=distance[nod[i]][nod[j]];
     }
   }
   for(i=0;i<(countsupplier+1);i++) //set coefficients of variablex in obj funt.
   {  for(j=0;j<(countsupplier+1);j++)
      { if(i<j)
                       { objective.setCoef(variablex[i][j],distancecoef[i][j]);
                       }
      }
   }
   for(i=0;i<(countsupplier+1);i++)//set coefficients of variablex in degree con
   {  for(j=0;j<(countsupplier+1);j++)
      { if(i<j)
        { degreeconstraint[i].setCoef(variablex[i][j],1);
        }
        if(j<i)
        { degreeconstraint[i].setCoef(variablex[j][i],1);
        }
      }
   }
   cplex.extract(model);
   subtour=1;
   while(subtour==1)
   {  cplex.solve();
      for(m=0;m<(countsupplier+1);m++)
              {         insubtour[m]=0; // to check if nod m is in other subtours already considered.
              }
      countnod=0; firstnod=0; totalnod=0; endnod=0; numbersubtour=0;
   findsubtour:numbersubtour++;
              for(j=0;j<(countsupplier+1);j++)
      {if(firstnod<j)
      { if(cplex.getValue(variablex[firstnod][j])>=0.9)
        {  findroute:countnod++;
           totalnod++;
           subroute[countnod]=j;
        repeat:for(k=0;k<(countsupplier+1);k++)
           { if(j<k)
              { if(((cplex.getValue(variablex[j][k]))>=0.9)&&(k!=firstnod))
                {repeat2:totalnod++;
                  countnod++;
                                             subroute[countnod]=k;
                  firstnod=j;
                  if(k==endnod)  break;
                  j=k;
                  goto repeat;
                }
              }  //close if(j<k)
              if(k<j)
              { if(((cplex.getValue(variablex[k][j]))>=0.9)&&(k!=firstnod))
                {  goto repeat2;
                }
              }
           } // close repeat:for
           break;
        } //close if(cplex.getValue(variablex[firstnod][j])>=0.9)
      } //if(firstnod<j)
      if(j<firstnod)
      { if((cplex.getValue(variablex[j][firstnod]))>=0.9)
        {  goto findroute;
        }
      }
      }// close for(j)
```

```
    if(countnod<countsupplier) // add the subtour elimination constraint.
    { for(i=0;i<(countsupplier+1);i++)
      {  checknod[i]=0;
      }
      for(i=1;i<=countnod;i++) // identify suppliers in the subtour.
      {  checknod[subroute[i]]=1;
                      insubtour[subroute[i]]=1;
      }
      IloExpr subtourconstraint(env);
      for(i=0;i<(countsupplier+1);i++)
      {  for(j=0;j<(countsupplier+1);j++)
         {  if(i<j)
           { if(((checknod[i]==1)&&(checknod[j]==0))||((checknod[i]==0)&&(checknod[j]==1)))
             {  subtourconstraint += variablex[i][j];
             }
           }
         }
      }
      model.add(subtourconstraint>=2);
    } // close adding the subtour elimination constraint.
            if(totalnod<(countsupplier+1))
    { for(i=1;i<(countsupplier+1);i++) //find the first sup for next subtour.
      {  for(j=1;j<=countnod;j++)
         {  if(i==subroute[j])  break;
           if((j==countnod)&&(insubtour[i]==0))
           {  firstnod=i;
              countnod=0;
              endnod=i;
              goto findsubtour;
           }
         }
      }
    }
    else
    { if(numbersubtour==1)  {
                                  subtour=0; // obtain the optimal tour.
                      }
    }
            //if(countsupplier==11) cin>>n;
  } //close while loop.
  cout<<"The Optimal TSP TOUR of vehicle: ";
  mintourlength=distancecoef[subroute[countsupplier+1]][subroute[1]];
  cout<<nod[subroute[countsupplier+1]]<<"-";
  for(i=1;i<=countsupplier;i++)
  {  mintourlength=mintourlength+distancecoef[subroute[i]][subroute[i+1]];
    cout<<nod[subroute[i]]<<"-";
  }
  cout<<subroute[countsupplier+1]<<endl;;
  env.end();
  delete[] subroute;
  delete[] insubtour;
  for(i=0;i<(countsupplier+1);i++)
 {  delete[]distancecoef[i];
 }
 delete[] distancecoef;
}
//*****************************************************************
```

โปรแกรม C++ Main Program แนวทางที่ 2 วิธี GRASP with VLSNสำหรับปัญหาVRP

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iomanip>
#include <fstream.h>
#include <sys/time.h>
#include <stdio.h>
/* -- Declaration of the procedure FindCycle and the structures used in this procedure */
using namespace std;
struct _ImpGraph {
        int nNodes;             /* -- Number of nodes in the improvement graph */
        int nSets;              /* -- Number of subsets in the improvement graph */
        float **aCost;   /* -- aCost[i][j] is the cost of arc from i to j */
        int *aSet;              /* -- aSet[i] is the subset to which node i belongs */
};
using namespace std;
void buildroute(int*&routebuild,int**customervehicle,int*numbercustomerin,int group);
void caldistance( float*& locatX, float*& locatY,float max, float min,
   float**& distance, int numbernode);
void randomassign(int**&variable, int numbervariable, int numberitem, int firstvariable);
void insertion(int*& routeinst, float** distance, float max, int numbernode,int numbercustomer);
void twoopt(int**&edge,float** distance,int numbernode,int*& routeinst,
   int**&customervehicle,int*numbercustomerin,int rvehicle);
float totaldistance(float** distance,int* route, int numbernode);
float ttcost(float*TSPcost,int numbervehicle);
void removecustomer(int**&edge,int**&customervehicle,int*&numbercustomerin,
   int movefromvehicle,int ordercustomerout,int customerout);
void addcustomer(int**&edge,float**distance,int**&customervehicle,
   int*&numbercustomerin,int movefromvehicle,int movetovehicle,
   int customerout,int numbercustomer,int insert1,int insert2);
void updateinsertion(int**&edge,float**distance,int*&numbercustomerin,
   int**&customervehicle,int select,int rvehicle,int numbercustomer,
   int insert1,int insert2);
float distanceinsert(float**distance,int*numcustomerin,int**customervehicle,
    int customerout,int k,int numbercustomer,int&tempinsert1,int&tempinsert2);
void onemovelocalsearch(int**&edge,float**distance,int*&numbercustomerin,
 int**&customervehicle,float*&totaldemandrate,float*&TSPcost,
 float*demandrate,float&TOTALCOST,int numbervehicle,int numbercustomer,
 float cap);
void customerexchange(int**&edge,float**distance,int*&numbercustomerin,
 int**&customervehicle,float*&totaldemandrate,
 float*&TSPcost,float*demandrate,float&TOTALCOST,int numbervehicle,
 int numbercustomer,float cap);
//****************************VLSN*****************************************
//*************************************************************************
void
buildgraph(int*&temcustomer,int*&vehicleoftemcustomer,float**&temtotaldemandrate,float**&te
mTSP
   cost,
float**&newcost,int**&teminsert1,int**&teminsert2,int**&checkin,int**&ordercustomerout,
   float**distance,int*numbercustomerin,int**customervehicle,float*totaldemandrate,
   float*TSPcost,int numbervehicle,int numbercustomer,float cap,
   float*demandoftemcustomer,
int*temroute1,int*temnumbercustomerin,int**temcustomervehicle,
   double infinity,int numbertemcustomer,struct _ImpGraph *ImpGraph);
void arcsuptosup(int i,int
j,int*&temcustomer,int*&vehicleoftemcustomer,float**&temtotaldemandrate,
   float**&temTSPcost,float**&newcost,int**&teminsert1,int**&teminsert2,int**&checkin,
   int**&ordercustomerout,float**distance,int*numbercustomerin,int**customervehicle,
   float*totaldemandrate,float*TSPcost,int numbercustomer,float cap,float*demandoftemcustomer,
   double infinity,int*temnumbercustomerin,int**temcustomervehicle,int*temroute1,
   int numbertemcustomer,struct _ImpGraph *ImpGraph);
void arcsuptovehicle(int i,int k,int*&temcustomer,float**&temtotaldemandrate,
   float**&temTSPcost,
   float**&newcost,int**&teminsert1,int**&teminsert2,int**&checkin,float**distance,
   int*numbercustomerin,int**customervehicle,int*&vehicleoftemcustomer,float*totaldemandrate,
   float*TSPcost,int numbercustomer,float cap,float*demandoftemcustomer,
   double infinity,int numbertemcustomer,struct _ImpGraph *ImpGraph);
void arcdummytosup(int
i,int*&temcustomer,int*&vehicleoftemcustomer,float**&temtotaldemandrate,
   float**&temTSPcost,float**&newcost,int**&ordercustomerout,float**distance,
   int*numbercustomerin,int**customervehicle,float*totaldemandrate,float*TSPcost,
   float cap,float*demandoftemcustomer,struct _ImpGraph *ImpGraph);
void updateimprovegraph(int
numnodeincycle,int*&temcustomer,int*&vehicleoftemcustomer,float**&temtotaldemandrate,
   float**&temTSPcost,float**&newcost,
   int**&teminsert1,int**&teminsert2,int**&checkin,int**&ordercustomerout,float**distance,
   int*&numbercustomerin,int**&customervehicle,float*&totaldemandrate,
   float*&TSPcost,int**&edge,int numbervehicle,int numbercustomer,float cap,
   float*demandrate,int*vehicle,int*aCycle,int*vehicleinvolve,double infinity,
   int*temnumbercustomerin,int**temcustomervehicle,int*temroute1,struct _ImpGraph
*ImpGraph,
   int&numbertemcustomer,float*&demandoftemcustomer);
static void MemStructInit(struct _MemStruct *pMem);
static void MemStructDestroy(struct _MemStruct *pMem);
static struct _DPState *DPStateAlloc(struct _MemStruct *pMem);
static void HashTableInit( struct _HashTable *pHash);
static void HashInsert(struct _HashTable *pHash, struct _DPState *pState);
static struct _DPState *HashFind(struct _HashTable *pHash, int i, unsigned long Set[], short stage);
int StateCmp(const void *p1, const void *p2);
/* -- Returns the length of a -ve cost subset disjoint cycle if found, otherwise -1,
  -- the cycle is returned in the array aCycle. The array aCycle contains
  -- the actual cycle as aCycle[0]<-aCycle[1]<-...<-aCycle[k] where aCycle[k] = aCycle[0] */
int FindCycle(struct _ImpGraph *ImpGraph, int *aCycle);
//***************************MAIN FUNCTION***********************************
//cap = vehicle capacity.
//countcustomer = counter of number of suppliers.
//countcustomeritem = counter of number of items produced by that customer.
//demandrate[i]= a demand rate of item i.
//distance[i][j]= an array of distance between node i and j.
//edge[i][j] = 1 if node i is connected with node j or 0, otherwise.
//groupout = a group (a vehicle) from which a customer is removed.
//groupin = a group (a vehicle) which a customer is added.
//holdcost[i] = a holding cost rate of item i.
```

```
//i,ii,iii,j,jj,k,m = counter for looping.
//item[i]= item i.
//max = the highest value of the coordinates x and y that is used to generate
//     the location of a node.
//min = the lowest value of the coordinates x and y that is used to generate
//     the location of a node.
//maxdemandrate = maximum demand rate.
//mindemandrate = minimum demand rate.
//maxholdcost = maximum inventory holding cost.
//minholdcost = minimum inventory holding cost.
//node[i] = a temporary array to hold customers.
//numberitem = number of items in the system.
//numbercustomer = number of customers in the system plus a warehouse.
//numbercustomerin[i]= number of customers visited by a vehicle i.
//numbervehicle = number of vehicles in the system.
//quantity[i]= replenishment quantity of item i.
//rangeholdcost = number of inventory holding cost rates in the range.
//rvehicle = a vehicle (a group) currently considered.
//rangedemand = number of demand rates in the range.
//routeinst[i]= an array of sequence of customers and a warehouse in a route
//          generated by the arbitrary insertion heuristic.
//routebuild[i]= an array of sequence of customers and a warehouse in a route
//          used in the totaldistance function.
//seed = a variable used to initialize the sequence of pseudo-random numbers.
//customer[i]= customer i.
//customeritem[i][j] = 1 if customer i produces item j or 0 otherwise.
//customerout = a customer who is removed from a group and is added to another.
//customervehicle[i][j]= a customer visited by a vehicle i in the order j.
//time[i] = replenishment interval of items picked up by vehicle i.
//totaldemandrate[i]= sum of demand rates of items picked up by vehicle i.
//totalholdcost[i] = an aggregate inventory holding cost for items in vehicle i.
//totalQ[i]= an aggregate replenishment quantity picked up vehicle i.
//totalcost[i] = average total inventory and transportation cost of items in
//          vehicle i.
//TOTALCOST = average total inventory and transportation cost of the system.
//TSPcost[i] = transportation cost( total travel distance ) of vehicle i.
//ttempitem[i] = a temporary array to hold items.
//vehivle[i] = vehicle i.
//vehicleitem[i][j]= 1 if vehicle i picks up item j or 0 otherwise.
int main()
{clock_t start,startV,startG;
 clock_t end,endV,endG;
 unsigned seed = time(NULL) ;
 srand(seed);
struct _ImpGraph *dataG;
//*******************VLSN VARIABLES****************************
 double infinity;
 float cyclecost;
 int numnodeincycle,numbertemcustomer,counttemcustomer,countiteration,
    countimprove;
 int*temcustomer,*vehicleoftemcustomer,*aCycle,
   *vehicleinvolve,*temroute1,*temnumbercustomerin;
 float*demandoftemcustomer;
 float**temtotaldemandrate,**temTSPcost,**newcost;
 int**teminsert1,**teminsert2,**checkin,**ordercustomerout,
   **temcustomervehicle;
//**************************************************************
 float cap,mindemandrate,maxdemandrate,diff,
      maxtotaldemandrate,minplus,min,max,TOTALCOST,smalltime,alpha,newalpha,
      BESTTOTALCOST,mintime,templus,mplus,TOTALCOST_VLSN;
 int countcustomer,maxfrequency,numbervehicle,numbercustomer,rvehicle,groupout,
    groupin,rangedemand,customerout,numbervehicleuse,
    i,ii,iii,j,jj,k,m,n,v,xyz=1,maxnumberrun,numberrun,vehicleuse,
    bestnumbervehicleuse,initialnumbervehicle,firstcust,countcust,custpick,
    countnumbercustomer,countcustinlist,countchange,bestnumberrun;
 int*customer,*custgroup,*routeinst,*routebuild,*vehicle,*node,
    *numbercustomerin,*bestnumbercustomerin,*custlist;
 float*demandrate,*quantity,*time,*totaldemandrate,*TSPcost,*totalQ,*totalcost,
     *besttotaldemandrate,*bestTSPcost,*besttotalQ,*besttotalcost,*plus;
 float**distance;
 int**edge,**customervehicle,**vehiclecust,**bestedge,**bestcustomervehicle,
     **bestvehiclecust;
 double diffVLSN;
 int *ttempcust;
 int**checkout;
 ifstream readfile;
 ofstream outputfile,outputfile1,outputfile2;
 while(xyz!=0)
{cout << "seed =" << seed << endl;
 cout<<endl<<"Total number of customers excluding a warehouse: ";
 cin>>numbercustomer;
 cout<<endl<<"Total number of vehicles: ";
 cin>>numbervehicle;
 cout<<endl<<"Vehicle capacity: ";
 cin>>cap;
 cout<<endl<<"Restricted candidate parameter: "; // 50% = 1.50
 cin>>alpha;
 cout<<endl<<"Maximum number of runs: ";        // 80,50 etc.
 cin>>maxnumberrun;
 distance = new (float*)[numbercustomer+1];
 for(ii=0; ii<=numbercustomer; ii++){
     distance[ii] = new (float)[numbercustomer+1];
 }
 cout.precision(5);
 cout<<endl;
//customer[0] represents the warehouse while customer[1] is Supplier1 and so on.
```

```
//Generate matrix of distance between nodes.
 int xxyy;
 //cin>>xxyy;
 custgroup = new int[numbercustomer+1];
 customer = new int[numbercustomer+1];
 ttempcust = new int[numbercustomer+1];
 custlist  = new int[numbercustomer+1];
 for(i=0;i<=numbercustomer;i++){
   customer[i]= i;   //customer[0] is the warehouse.
 }
 vehicle = new int[numbervehicle];
 for(i=0;i<numbervehicle;i++){
   vehicle[i]= i;
 }
 routebuild = new int[numbercustomer+1];
 numbercustomerin = new int[numbervehicle];
 bestnumbercustomerin = new int[numbervehicle];
 node = new int[numbercustomer+1];
 time = new float[numbervehicle];
 totaldemandrate = new float[numbervehicle];
 TSPcost = new float[numbervehicle];
 totalQ = new float[numbervehicle];
 totalcost = new float[numbervehicle];
 besttotaldemandrate = new float[numbervehicle];
 bestTSPcost = new float[numbervehicle];
 besttotalQ = new float[numbervehicle];
 besttotalcost = new float[numbervehicle];
 plus = new float[numbercustomer+1];
 demandrate = new float[numbercustomer+1];
 edge = new (int*)[numbercustomer+1];
 for(i=0;i<=numbercustomer;i++){
   edge[i]= new (int)[numbercustomer+1];
 }
 vehiclecust = new (int*)[numbervehicle];
 bestvehiclecust = new (int*)[numbervehicle];
 customervehicle = new (int*)[numbervehicle];
 for(i=0;i<numbervehicle;i++){
   customervehicle[i]= new (int)[numbercustomer+1];
   vehiclecust[i] = new (int)[numbercustomer+1];
   bestvehiclecust[i] = new (int)[numbercustomer+1];
 }
 for(i=0;i<numbervehicle;i++)
 {    for(k=0;k<=numbercustomer;k++)
        customervehicle[i][k]= 0;
        vehiclecust[i][k] = 0;
 }
 bestedge = new (int*)[numbercustomer+1];
 for(i=0;i<=numbercustomer;i++){
   bestedge[i]= new (int)[numbercustomer+1];
 }
 bestcustomervehicle = new (int*)[numbervehicle];
 for(i=0;i<numbervehicle;i++){
   bestcustomervehicle[i]= new (int)[numbercustomer+1];
 }
 initialnumbervehicle=numbervehicle;
 int tt;
  int loop=0;
 diffVLSN=0.0;
 double diffG=0.0;
 int countimprove=0;
 readfile.open("waste_358_collecting_points.out"); // Data of Ubon waste collection-There are 358
points and 1 depot.
 outputfile.open("result_GRASP_VLSN_Waste.out");
while(loop<1)
{for( i=0; i<=numbercustomer; i++)
 {  for( j=0; j<=numbercustomer; j++)
   {   readfile>>distance[i][j];
   }
 }
 for( i=1; i<=numbercustomer; i++)
 {  readfile>>demandrate[i];
 }
//***********************************heuristic*********
 start=clock();
 startG=clock();
 startV=clock();
 mintime=1000.0;
 float mindistance=1000000;   // new DS ******ON fEB 9 *****************
// start=clock(); // starting measuring run time.
 for( i=1; i<=numbercustomer; i++)
 {  if(distance[0][i]<mindistance)
   {  firstcust = i;
      mindistance = distance[0][i];
   }
   custgroup[i] = 0;
 }
 numberrun=0;
 BESTTOTALCOST=10000000.0;
 //perform construction phase and local search = maxnumberrun times.
 while(numberrun<maxnumberrun)
 {  newalpha=alpha;
   countchange=0;
   numbervehicleuse = 1;
   vehicleuse=0;
   vehiclecust[vehicleuse][firstcust]=1;
   custgroup[firstcust]=1;
   totaldemandrate[vehicleuse]= demandrate[firstcust];
```

```
    custpick=firstcust;
    countnumbercustomer = 1;
    countcust=1;
    ttempcust[countcust]=custpick;  //ttempcust[1] is the first one.
construct:while(countnumbercustomer<numbercustomer)   //iteration of the construction phase.
    {   minplus = 10000.0;
       for(i=1;i<=numbercustomer;i++)
       {  if(custgroup[i]==0)
         {   mplus=1000;
           for(j=1;j<=countcust;j++)
           {   //cout<<" er1"<<endl;
             templus=distance[ttempcust[j]][i]/distance[0][i];
             //cout<<" er2"<<endl;
             if(templus<mplus)
             {  mplus=templus;
               plus[i]=templus;
             }
           }
           if(plus[i]<minplus)
           {  minplus=plus[i];
           }
         }
       }
       countcustinlist=0;
       for(i=1;i<=numbercustomer;i++)
       {  if((plus[i]<=(newalpha*minplus))&&(custgroup[i]==0))
         {  custlist[countcustinlist]=i;
           countcustinlist++;
         }
       }
       if(countcustinlist>4)          //  add on May 23,2012 to limit the # of customers = 3
       {   custpick=custlist[rand()%4];
       }
       else
       {
          custpick=custlist[rand()%countcustinlist];
       }
       if((totaldemandrate[vehicleuse]+demandrate[custpick])<=cap)
       {   totaldemandrate[vehicleuse]=totaldemandrate[vehicleuse]+demandrate[custpick];
          countcust++;
       }
       else
       {   vehicleuse++; // use a new vehicle.
          numbervehicleuse++;
          countcust=1;
          mindistance=1000000;
          for(i=1;i<=numbercustomer;i++)
          {   if(custgroup[i]==0)
             {  if(distance[0][i]<mindistance)
               {  custpick=i;
                 mindistance=distance[0][i];
```

```
            }
          }
        }
        totaldemandrate[vehicleuse]= demandrate[custpick];
        ttempcust[countcust]=custpick;
        custgroup[custpick]= 1;
        countnumbercustomer++;
        vehiclecust[vehicleuse][custpick]= 1;
        newalpha=alpha; // Add on Sept 03,2008 to return alpha to original one ******
        goto construct;
      }
      ttempcust[countcust]=custpick;
      custgroup[custpick]= 1;
      countnumbercustomer++;
      vehiclecust[vehicleuse][custpick]= 1;  //assign a customer to a vehicle.
    } //close while loop of iteration of the construstion phase.
    numbervehicle=numbervehicleuse;
//For each vehicle, compute the total distance traveled.
    for(i=0;i<numbervehicle;i++)
    {   countcustomer = 0;
      node[0] = 0;
      for(ii=1;ii<=numbercustomer;ii++){
        node[ii]= ii;
      }
      for(iii=1;iii<=numbercustomer;iii++)
      {   if(vehiclecust[i][iii]==1)
        {     countcustomer++;
              node[countcustomer]= iii;
        }
      }
      numbercustomerin[i]= countcustomer;
      routeinst = new int[500];   // increase from 40 to 500 for waste
      for(m=0;m<(countcustomer+1);m++) {
        routeinst[m]= node[m];
      }
      insertion(routeinst,distance,max,(countcustomer+1),numbercustomer+1);
      TSPcost[i] = totaldistance(distance,routeinst,(countcustomer+1));
      twoopt(edge,distance,(countcustomer+1),routeinst,customervehicle,
    numbercustomerin,i);
      for(m=0;m<=numbercustomerin[i];m++) {
        cout<<customervehicle[i][m]<<"-";
      }
      cout<<endl;
      TSPcost[i] = totaldistance(distance,routeinst,(countcustomer+1));
      delete[]routeinst;
    }
    TOTALCOST = ttcost(TSPcost,numbervehicle);
    endV=clock();
    diffVLSN=diffVLSN+(endV-startV)/CLOCKS_PER_SEC;
//********************VLSN*************************************************
//**************************************************************************
```

```
//******Start counting nodes for Supplier VLSN******************************
 //** start=clock();
 numbertemcustomer=0;
 for(i=0;i<numbervehicle;i++)
 {  numbertemcustomer=numbertemcustomer+numbercustomerin[i];
 }  // consider the same customer as a different node if in diferent vehicles.
 dataG->nNodes=numbertemcustomer+numbervehicle+1;
//*********************IMPROVEMENT GRAPH VARIABLES**************************
 vehicleinvolve=new int[numbervehicle];
 aCycle=new int[dataG->nNodes];
 temcustomer=new int[numbertemcustomer];
 vehicleoftemcustomer=new int[numbertemcustomer];
 demandoftemcustomer=new float[numbertemcustomer];
 temtotaldemandrate=new (float*)[dataG->nNodes];
 temTSPcost=new (float*)[dataG->nNodes];
 dataG->aCost=new (float*)[dataG->nNodes];
 newcost=new (float*)[dataG->nNodes];
 teminsert1=new (int*)[dataG->nNodes];
 teminsert2=new (int*)[dataG->nNodes];
 checkin=new (int*)[dataG->nNodes];
 ordercustomerout=new (int*)[dataG->nNodes];
 for(i=0;i<dataG->nNodes;i++)
 {  temtotaldemandrate[i]=new float[dataG->nNodes];
    temTSPcost[i]=new float[dataG->nNodes];
    dataG->aCost[i]=new float[dataG->nNodes];
    newcost[i]=new float[dataG->nNodes];
    teminsert1[i]=new int[dataG->nNodes];
    teminsert2[i]=new int[dataG->nNodes];
    checkin[i]=new int[dataG->nNodes];
    ordercustomerout[i]=new int[dataG->nNodes];
 }
 dataG->aSet=new int[dataG->nNodes];
 temroute1=new int[numbercustomer];
 temcustomervehicle=new (int*)[numbervehicle];
 temnumbercustomerin=new int[numbervehicle];
 for(i=0;i<numbervehicle;i++)
 {  temcustomervehicle[i]=new (int)[numbercustomer];
 }
dataG->nSets=numbervehicle+1;
counttemcustomer=0;
for(i=0;i<numbervehicle;i++)
{  dataG->aSet[numbertemcustomer+i]=i;
   for(j=1;j<=numbercustomerin[i];j++)
   {  temcustomer[counttemcustomer]=customervehicle[i][j];
      vehicleoftemcustomer[counttemcustomer]=i;
      dataG->aSet[counttemcustomer]=i;
      demandoftemcustomer[counttemcustomer]=demandrate[temcustomer[counttemcustomer]];
      counttemcustomer++;
   }
}
dataG->aSet[dataG->nNodes-1]=numbervehicle; // Assign a set to the dummy node.
infinity=1000000.0;
```

```
buildgraph(temcustomer,vehicleoftemcustomer,temtotaldemandrate,
   temTSPcost,newcost,teminsert1,teminsert2,checkin,ordercustomerout,distance,
   numbercustomerin,customervehicle,totaldemandrate,
   TSPcost,numbervehicle,numbercustomer,cap,
   demandoftemcustomer,temroute1,temnumbercustomerin,temcustomervehicle,
   infinity,numbertemcustomer,dataG);
numnodeincycle=FindCycle(dataG,aCycle);
//**cout<<"numnodeincycle = "<<numnodeincycle<<endl;
countiteration=0;
while(numnodeincycle>0)
{updateimprovegraph(numnodeincycle,temcustomer,vehicleoftemcustomer,
  temtotaldemandrate,temTSPcost,newcost,teminsert1,teminsert2,checkin,
  ordercustomerout,distance,numbercustomerin,customervehicle,
  totaldemandrate,TSPcost,edge,numbervehicle,numbercustomer,cap,
  demandrate,vehicle,aCycle,vehicleinvolve,infinity,
  temnumbercustomerin,temcustomervehicle,temroute1,
  dataG,numbertemcustomer,demandoftemcustomer);
 buildgraph(temcustomer,vehicleoftemcustomer,temtotaldemandrate,
   temTSPcost,newcost,teminsert1,teminsert2,checkin,ordercustomerout,distance,
   numbercustomerin,customervehicle,totaldemandrate,
   TSPcost,numbervehicle,numbercustomer,cap,
   demandoftemcustomer,temroute1,temnumbercustomerin,temcustomervehicle,
   infinity,numbertemcustomer,dataG);
 numnodeincycle=FindCycle(dataG,aCycle);
 countiteration++;
 cyclecost=0;
 for(n=numnodeincycle;n>0;n--)
 {  cyclecost=cyclecost+dataG->aCost[aCycle[n]][aCycle[n-1]];
 }
 if( cyclecost>=-0.01)
 { break;
 }
}
TOTALCOST_VLSN = ttcost(TSPcost,numbervehicle);

//************************ 2 opt after VLSN ***********************************
//For each vehicle, compute the total distance traveled.
  for(i=0;i<numbervehicle;i++)
  {  routeinst = new int[500];  // increase to 500
      for(m=0;m<(numbercustomerin[i]+1);m++) {
         routeinst[m]= customervehicle[i][m];
      }
      twoopt(edge,distance,(numbercustomerin[i]+1),routeinst,customervehicle,
  numbercustomerin,i);
      for(m=0;m<=numbercustomerin[i];m++) {
         cout<<customervehicle[i][m]<<"-";
      }
      cout<<endl;
      TSPcost[i] = totaldistance(distance,routeinst,(numbercustomerin[i]+1));
      delete[]routeinst;
  }
```

```
//********************** End 2 opt after VLSN ********************************
//*****************************************************************************
 TOTALCOST = ttcost(TSPcost,numbervehicle);
 if(countiteration!=0)
 {  countimprove++;
 }
 outputfile<<TOTALCOST<<"  "<<TOTALCOST_VLSN<<endl;  // print cost before 2opt too
 outputfile1<<countiteration<<endl;
//*********************END VLSN*********************************************
   if(TOTALCOST<BESTTOTALCOST)      //Update the best solution obtained from each run
   {   BESTTOTALCOST=TOTALCOST;
       bestnumbervehicleuse=numbervehicle;
       bestnumberrun=numberrun;
       for(i=0;i<bestnumbervehicleuse;i++)
       {  besttotaldemandrate[i]=totaldemandrate[i];
          bestTSPcost[i]=TSPcost[i];
          bestnumbercustomerin[i]=numbercustomerin[i];
          for(k=0;k<=bestnumbercustomerin[i];k++)
          {  bestcustomervehicle[i][k]=customervehicle[i][k];
          }
          for(j=0;j<=numbercustomer;j++)
          {  for(m=0;m<=numbercustomer;m++)
             {  bestedge[j][m]=edge[j][m];
             }
          }
       }
   }  // close update.
   numberrun++;
   if(numberrun<maxnumberrun)
   {  for(i=0;i<numbervehicle;i++)
     { for(j=0;j<=numbercustomer;j++)
      {   vehiclecust[i][j]=0;
      }
      for(k=0;k<=numbercustomer;k++)
      {   customervehicle[i][k]=0;
          for(ii=0;ii<=numbercustomer;ii++)
          {  edge[k][ii]=0;
          }
      }
     }
     for(m=0;m<=numbercustomer;m++)
     {  custgroup[m]=0;
     }
     //*******************VLSN****************************************
     delete[]temcustomer;
     delete[] vehicleoftemcustomer;
     delete[] demandoftemcustomer;
     for(i=0;i<numbervehicle;i++)
     {  delete[]temcustomervehicle[i];
     }
     delete[]temcustomervehicle;
```

```
      for(i=0;i<dataG->nNodes;i++)
      {   delete[]temtotaldemandrate[i];
          delete[]temTSPcost[i];
          delete[]dataG->aCost[i];
          delete[]newcost[i];
          delete[]teminsert1[i];
          delete[]teminsert2[i];
          delete[]checkin[i];
          delete[]ordercustomerout[i];
      }
      delete[]temnumbercustomerin;
      delete[]temroute1;
      delete[]temtotaldemandrate;
      delete[]temTSPcost;
      delete[]dataG->aCost;
      delete[]newcost;
      delete[]teminsert1;
      delete[]teminsert2;
      delete[]checkin;
      delete[]dataG->aSet;
      delete[]ordercustomerout;
      delete[]vehicleinvolve;
      delete[]aCycle;
      //***********************************************************************
      numbervehicle=initialnumbervehicle;
    }
    //cin>> ff;
  }  // close number of construction phase and local search while loop
  end=clock();
  double diff=(end-start)/CLOCKS_PER_SEC;
  cout<<"Run time = "<<diff<<endl;
  numbervehicle=bestnumbervehicleuse;
  TOTALCOST=BESTTOTALCOST;
  endG=clock();
  diffG=diffG+(endG-startG)/CLOCKS_PER_SEC;
  for(i=0;i<bestnumbervehicleuse;i++)
  { if(bestTSPcost[i]!=0)
    { cout<<"TSP TOUR of vehicle "<<i<<" : ";
      for(j=0;j<=bestnumbercustomerin[i];j++)
      {  cout<<bestcustomervehicle[i][j]<<"-";
      }
      cout<<bestcustomervehicle[i][0]<<endl;
      cout<<endl<<"TSP cost of vehicle "<<i<<" : "<<bestTSPcost[i]<<endl;
      cout<<endl<<"Accumulated demands of customers in vehicle "<<i<<" :
"<<besttotaldemandrate[i]<<endl;
      cout<<"Total quantity picked up by vehicle "<<i<<" : "<<besttotaldemandrate[i]<<endl;
    }
    else
    { cout<<" TSP cost of vehicle "<<i<<" = "<<bestTSPcost[i]<<" and this vehicle is not used."<<endl;
    }
  }
  cout<<endl<<"Best total System Cost ontained from GRASP= "<<BESTTOTALCOST<<endl;
  outputfile<<TOTALCOST<<endl;
  cout<<endl<<"bestnumberrun="<<bestnumberrun;
  //cin>>v;
```

```
int ff;
//cin>>ff;
loop++;
if(loop<1)
{  for(i=0;i<numbervehicle;i++)
   { for(j=0;j<=numbercustomer;j++)
     {   vehiclecust[i][j]=0;
     }
     for(k=0;k<=numbercustomer;k++)
     {   customervehicle[i][k]=0;
         for(ii=0;ii<=numbercustomer;ii++)
         {  edge[k][ii]=0;
         }
     }
   }
   numbervehicle=initialnumbervehicle;
}
}  // close data set while loop.
cout<<"Average computational time of VLSN = "<<diffVLSN/10<<endl;
cout<<"Average computational time of GRASP = "<<diffG/10<<endl;
cout<<"number of problems improved = "<<countimprove<<endl;
cin>>v;
readfile.close();
outputfile.close();
//outputfile1.close();
//outputfile2.close();
for(i=0;i<=numbercustomer;i++)
{   delete[]distance[i];
    delete[]edge[i];
    delete[]bestedge[i];
}
for(i=0;i<numbervehicle;i++)
{   delete[]vehiclecust[i];
    delete[]customervehicle[i];
    delete[]bestvehiclecust[i];
    delete[]bestcustomervehicle[i];
}
delete[]temnumbercustomerin;
delete[]temroute1;
delete[]temtotaldemandrate;
delete[]temTSPcost;
delete[]newcost;
delete[]teminsert1;
delete[]teminsert2;
delete[]checkout;
delete[]checkin;
delete[]ordercustomerout;
delete[] custlist;
delete[] plus;
delete[] distance;
delete[] vehiclecust;
delete[] edge;
delete[] customervehicle;
```

```
 delete[] bestvehiclecust;
 delete[] bestedge;
 delete[] bestcustomervehicle;
 delete[] routebuild;
 delete[] numbercustomerin;
 delete[] bestnumbercustomerin;
 delete[] customer;
 delete[] ttempcust;
 delete[] vehicle;
 delete[] node;
 delete[] demandrate;
 delete[] quantity;
 delete[] time;
 delete[] totaldemandrate;
 delete[] TSPcost;
 delete[] besttotaldemandrate;
 delete[] bestTSPcost;
 delete[] custgroup;
 cin>>xyz;
}
 return 0;
}
//**************************************************************************
```