```
double m_valueMatrix[7][7];
double m_discountFactor;
double m_cost;


int m_acceptanceLevel[7];
BOOL m_matchingMatrix[7][7];

double clubfee;




m_valueMatrix[ 1 ][ 1 ] =   1;
m_valueMatrix[ 1 ][ 2 ] =   2;
m_valueMatrix[ 1 ][ 3 ] =   4;
m_valueMatrix[ 1 ][ 4 ] =   6;
m_valueMatrix[ 1 ][ 5 ] =   10;
m_valueMatrix[ 1 ][ 6 ] =   17;

m_valueMatrix[ 2 ][ 1 ] =   m_valueMatrix[ 1 ][ 2 ];
m_valueMatrix[ 2 ][ 2 ] =   6;
m_valueMatrix[ 2 ][ 3 ] =   10;
m_valueMatrix[ 2 ][ 4 ] =   14;
m_valueMatrix[ 2 ][ 5 ] =   27;
m_valueMatrix[ 2 ][ 6 ] =   38;

m_valueMatrix[ 3 ][ 1 ] =   m_valueMatrix[ 1 ][ 3 ];
m_valueMatrix[ 3 ][ 2 ] =   m_valueMatrix[ 2 ][ 3 ];
m_valueMatrix[ 3 ][ 3 ] =   25;
m_valueMatrix[ 3 ][ 4 ] =   39;
m_valueMatrix[ 3 ][ 5 ] =   57;
m_valueMatrix[ 3 ][ 6 ] =   74;

m_valueMatrix[ 4 ][ 1 ] =   m_valueMatrix[ 1 ][ 4 ];
m_valueMatrix[ 4 ][ 2 ] =   m_valueMatrix[ 2 ][ 4 ];
m_valueMatrix[ 4 ][ 3 ] =   m_valueMatrix[ 3 ][ 4 ];
m_valueMatrix[ 4 ][ 4 ] =   50;
m_valueMatrix[ 4 ][ 5 ] =   68;
m_valueMatrix[ 4 ][ 6 ] =   89;

m_valueMatrix[ 5 ][ 1 ] =   m_valueMatrix[ 1 ][ 5 ];
m_valueMatrix[ 5 ][ 2 ] =   m_valueMatrix[ 2 ][ 5 ];
m_valueMatrix[ 5 ][ 3 ] =   m_valueMatrix[ 3 ][ 5 ];
m_valueMatrix[ 5 ][ 4 ] =   m_valueMatrix[ 4 ][ 5 ];
```

```
	m_valueMatrix[ 5 ][ 5 ] =   90;
	m_valueMatrix[ 5 ][ 6 ] =   116;

	m_valueMatrix[ 6 ][ 1 ] =   m_valueMatrix[ 1 ][ 6 ];
	m_valueMatrix[ 6 ][ 2 ] =   m_valueMatrix[ 2 ][ 6 ];
	m_valueMatrix[ 6 ][ 3 ] =   m_valueMatrix[ 3 ][ 6 ];
	m_valueMatrix[ 6 ][ 4 ] =   m_valueMatrix[ 4 ][ 6 ];
	m_valueMatrix[ 6 ][ 5 ] =   m_valueMatrix[ 5 ][ 6 ];
	m_valueMatrix[ 6 ][ 6 ] =   150;




	m_discountFactor = 4.0/5.0;
	m_cost = 2;
	clubfee =0;


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
	CStdioFile outFile;
	outFile.Open("out.txt", CFile::modeCreate | CFile::modeReadWrite | CFile::shareDenyWrite );

	CString outString;
	outString.Format("Discount factor\t%f\n",(float) m_discountFactor);
	outFile.WriteString(outString);

	outString.Format("cost\t%f\n",(float) m_cost);
	outFile.WriteString(outString);


	const long numTypes = 6;
	const long numOfOneType = 2;
	const long queueSize = 2;// number of PAIRS
	const long numInMarket=numTypes * numOfOneType;
	const long numGroups= numInMarket/2;

	long i,j, i1,i2;

	for ( i1=1;i1<=numTypes;i1++) {
		outString.Format("\t%d",i1);
		outFile.WriteString(outString);
	}
```

```
for ( i1=1;i1<=numTypes;i1++) {
       outString.Format("\n%d",i1);
       outFile.WriteString(outString);
       for ( i2=1;i2<=numTypes;i2++) {
              outString.Format("\t%f",(float)m_valueMatrix[i1][i2]);
              outFile.WriteString(outString);
       }
}



for ( m_acceptanceLevel[1]= 1;m_acceptanceLevel[1]<=1;m_acceptanceLevel[1]++) {
for ( m_acceptanceLevel[2]= m_acceptanceLevel[1];m_acceptanceLevel[2]<=2;m_acceptanceLevel[2]++) {
for ( m_acceptanceLevel[3]= m_acceptanceLevel[2];m_acceptanceLevel[3]<=3;m_acceptanceLevel[3]++) {
for ( m_acceptanceLevel[4]= m_acceptanceLevel[3];m_acceptanceLevel[4]<=4;m_acceptanceLevel[4]++) {
for ( m_acceptanceLevel[5]= m_acceptanceLevel[4];m_acceptanceLevel[5]<=5;m_acceptanceLevel[5]++) {
for ( m_acceptanceLevel[6]= m_acceptanceLevel[5];m_acceptanceLevel[6]<=6;m_acceptanceLevel[6]++) {

for ( m_acceptanceLevel[1]= 1;m_acceptanceLevel[1]<=6;m_acceptanceLevel[1]++) {
for ( m_acceptanceLevel[2]= 1;m_acceptanceLevel[2]<=6;m_acceptanceLevel[2]++) {
for ( m_acceptanceLevel[3]= 1;m_acceptanceLevel[3]<=6;m_acceptanceLevel[3]++) {
for ( m_acceptanceLevel[4]= 1;m_acceptanceLevel[4]<=6;m_acceptanceLevel[4]++) {
for ( m_acceptanceLevel[5]= 1;m_acceptanceLevel[5]<=6;m_acceptanceLevel[5]++) {
for ( m_acceptanceLevel[6]= 1;m_acceptanceLevel[6]<=6;m_acceptanceLevel[6]++) {



       for (i=1;i<=6; i++) {
              for (j=1;j<=6; j++) {
                     m_matchingMatrix[i][j] = (m_acceptanceLevel[i]<=j )&& (m_acceptanceLevel[j]<=i );
              }
       }




       double AbsQueuePosProbability[numTypes+1][queueSize+1];
       double CondQueuePosProbability[numTypes+1][queueSize+1];
       double NumMatchesProbability[numGroups+1];

       double matchprobability[numTypes+1][numTypes+1];
       // [i][j] = probability that i matches j; [i][0]  = probability that i does not matches

       double distributionOfLeaver[numTypes+1];


       long number[numInMarket+1];
```

```
long pos;

for (i=1;i<=numInMarket;i++) {
	number[i]=0;
}

for (i=1;i<=numTypes;i++) {
	for (j=0;j<=queueSize;j++) {
		AbsQueuePosProbability[i][j]=0;
	}
}

for (i=0;i<=numGroups;i++) {
	NumMatchesProbability[i]=0;
}

pos=1;
long NumCases =0;
double pSum =0.0;

while (TRUE) {
	while (pos>0) {
		number[pos]++;
		if ( number[pos] <=6) {
			break;
		}
		else {
			number[pos]=0;
			pos=pos-1;
		}
	}
	if (pos==0) break;

	BOOL checkOK = TRUE;
	// check no triple
	long numSame =0;
	for (i=1;i<pos;i++) {
		if ( number[pos]==number[i] ) numSame++;
	}
	if ( numSame>=numOfOneType ) checkOK = FALSE;
	// check Group
	if ( checkOK ) {
		if ( IsEven( pos) ) {
			if ( number[pos]<number[pos-1] ) {
				checkOK = FALSE;
			}
			else if ( pos>=4) {
```

```
        // check to previous group
                    if(  number[pos-1]<number[pos-3] ){
                          checkOK = FALSE;
                    }
                    if(  (number[pos-1]==number[pos-3]) &&(number[pos]<number[pos-2] ) ){
                          checkOK = FALSE;
                    }
              }
        }
}

if (checkOK) {
        if ( pos == numInMarket ) {
              // display
              // calculate probability
              // number of equal groups
              long numEqualGroups = 0;
              long numHeterogeneousGroups = 0;
              for ( i = 1;i<=numTypes;i++ ) {
                    if ( number[2*i-1]!=number[2*i]) {
                          numHeterogeneousGroups ++;
                    }
                    for ( j = i+1;j<=numTypes;j++ ) {
                          if ( (number[2*i-1]==number[2*j-1])&&(number[2*i]==number[2*j])) {
                                numEqualGroups++;
                          }
                    }
              }

              double counter =1;
              for (i=1;i<=6+numHeterogeneousGroups-numEqualGroups;i++) {
                    counter *=2.0;
              }

              double probability = counter/(7.0*8.0*9.0*10.0*11.0*12.0);
              pSum+= probability;

              long numMatchers[numTypes+1];
              for (i=1;i<=numTypes;i++) {
                    numMatchers[i]=0;
              }
              long numMatches =0;

              for (i=1;i<=numGroups;i++) {
                    long a= number[2*i-1];
                    long b= number[2*i];
                    if ( m_matchingMatrix[ a][b]) {
```

```
                                        numMatchers[a]++;
                                        numMatchers[b]++;
                                        numMatches++;
                                  }
                            }


                            double ProbQueuePosOfLeaver[queueSize+1];
                            ProbQueuePosOfLeaver[0] = (numMatches<=queueSize) ? 0.0:
                                                                  (0.0+numMatches-queueSize)/numMatches;
                            for (j=1;j<=queueSize;j++ ) {
                                  ProbQueuePosOfLeaver[j] = j+numMatches <= queueSize ?0.0: 1.0/numMatches;
                            }

                            for (j=0;j<=queueSize;j++ ) {
                                  for ( i=1;i<=numTypes;i++) {
                                        AbsQueuePosProbability[i][j] += probability * ProbQueuePosOfLeaver[j] *
numMatchers[i]/numOfOneType;
                                  }
                            }
                            NumMatchesProbability[numMatches]+=probability;


                      }
                      else {
                            pos ++;
                      }
                }

          }

          for ( i=1;i<=numTypes;i++) {
                double totalprobability =0;
                for (j=0;j<=queueSize;j++ ) {
                      totalprobability+= AbsQueuePosProbability[i][j];
                }
                for (j=0;j<=queueSize;j++ ) {
                      CondQueuePosProbability[i][j]=AbsQueuePosProbability[i][j]/totalprobability;
                }
          }

          double factorOfQueue[numTypes+1]; // factor with which the rebirth value has to be multiplied
          for ( i=1;i<=numTypes;i++) {
                factorOfQueue[i]=0;
          }

          for ( i=1;i<=numTypes;i++) {
```

```
	double QueuePosProbability[queueSize+1];
	double NewQueuePosProbability[queueSize+1];
	for (j=0;j<=queueSize;j++ ) {
		QueuePosProbability[j]= CondQueuePosProbability[i][j];
	}

	double factor=m_discountFactor;
	long counter =20;
	while ( counter -->=0){
		factorOfQueue[i]+= factor * QueuePosProbability[0];
		factor *= m_discountFactor;
		QueuePosProbability[0]=0;

		for (j=0;j<=queueSize;j++ ) {
			NewQueuePosProbability[j]= 0.0;
		}
		long k;
		for (k=0;k<=numGroups;k++ ) {
			for (j=0;j<=queueSize;j++ ) {
				long newpos= j-k;
				if (newpos<0) newpos = 0;
				NewQueuePosProbability[newpos]+=QueuePosProbability[j]*NumMatchesProbability[k];
			}
		}
		for (j=0;j<=queueSize;j++ ) {
			QueuePosProbability[j]= NewQueuePosProbability[j];
		}
	}
}



long i1,i2;
double sumProbMatch=0;
for ( i1=1;i1<=numTypes;i1++) {
	double probabilityOfNoMatch =1;
	for ( i2=1;i2<=numTypes;i2++) {
		if (m_matchingMatrix[i1][i2]) {
			double numChances= numOfOneType;
			if ( i1==i2 ) numChances= numOfOneType-1;
			matchprobability[i1][i2]= numChances/(numInMarket-1);
			probabilityOfNoMatch -= numChances/(numInMarket-1);
		}
		else {
			matchprobability[i1][i2]=0;
		}
```

```
	}
	matchprobability[i1][0]= probabilityOfNoMatch;
	sumProbMatch +=1-probabilityOfNoMatch;
}

for ( i=1;i<=numTypes;i++) {
	distributionOfLeaver[i] = (1-matchprobability[i][0])/sumProbMatch;
}


double Values[numTypes+1];
for ( i=1;i<=numTypes;i++) {
	Values[i]=0;
}
double NewValues[numTypes+1];

double ValueOfBirth=0;
long counter = 100;
while (counter-- >0) {
	ValueOfBirth=0;
	for ( i=1;i<=numTypes;i++) {
		ValueOfBirth += distributionOfLeaver[i]*Values[i];
	}


	for ( i1=1;i1<=numTypes;i1++) {
		NewValues[i1]=-m_cost+matchprobability[i1][0]*Values[i1]*m_discountFactor;
		for ( i2=1;i2<=numTypes;i2++) {
			NewValues[i1] += matchprobability[i1][i2]*m_valueMatrix[i1][i2];
		}
		NewValues[i1]+=(1-matchprobability[i1][0])*factorOfQueue[i1]*ValueOfBirth;
	}

	for ( i=1;i<=numTypes;i++) {
		Values[i]=NewValues[i];
	}
}

BOOL ExPostAcceptanceMatrix[numTypes+1][numTypes+1];
BOOL allOK=TRUE;
for ( i1=1;i1<=numTypes;i1++) {
	for ( i2=1;i2<=numTypes;i2++) {
		double valueAccept = -m_cost + m_valueMatrix[i1][i2] +factorOfQueue[i1]*ValueOfBirth;
		double valueReject = -m_cost+m_discountFactor*Values[i1];
		ExPostAcceptanceMatrix[i1][i2]=valueAccept>=valueReject;
		if (( i1>=i2 )&& (ExPostAcceptanceMatrix[i1][i2]!=m_matchingMatrix[i1][i2])){
			allOK=FALSE;
```

```
            }
        }
}
if (allOK) {


        outString.Format("\nAcceptance",i1);
        outFile.WriteString(outString);

        for ( i1=1;i1<=numTypes;i1++) {
              outString.Format("\t%d",m_acceptanceLevel[i1]);
              outFile.WriteString(outString);
        }

        outString.Format("\nRebirthValue\t%f",ValueOfBirth);
        outFile.WriteString(outString);

        outString.Format("\ndistribution of leavers");
        outFile.WriteString(outString);

        for ( i1=1;i1<=numTypes;i1++) {
              outString.Format("\t%f",(float)distributionOfLeaver[i1]);
              outFile.WriteString(outString);
        }

        outString.Format("\nValues");
        outFile.WriteString(outString);

        for ( i1=1;i1<=numTypes;i1++) {
              outString.Format("\t%f",(float)Values[i1]);
              outFile.WriteString(outString);
        }


        outString.Format("\nfactor of queue");
        outFile.WriteString(outString);

        for ( i1=1;i1<=numTypes;i1++) {
              outString.Format("\t%f",(float)factorOfQueue[i1]);
              outFile.WriteString(outString);
        }


        outString.Format("\nmatching probability");
        for ( i1=1;i1<=numTypes;i1++) {
              outString.Format("\n%d",i1);
              outFile.WriteString(outString);
```

```
                        for ( i2=1;i2<=numTypes;i2++) {
                               outString.Format("\t%f",(float)matchprobability[i1][i2]);
                               outFile.WriteString(outString);
                        }
                 }


          }

       }}}}}}




       //Perfect costless club
       long firstInClub=0;
       for ( firstInClub= 2;firstInClub<=numTypes;firstInClub++) {
              long numTypesOutOfClub = firstInClub-1;
              long numTypesInClub = numTypes-numTypesOutOfClub;

              double matchprobability[numTypes+1][numTypes+1]; // [i][j] = probability that i matches j; [i][0]  = probability
that i does not matches
              double distributionOfLeaver[numTypes+1];


              double clubFactorOfQueue=
                           (0.0+numGroups-queueSize)/numGroups * m_discountFactor // those who are the first time out
                           +(0.0+queueSize)/numGroups * m_discountFactor*m_discountFactor; // second time out

              long i1,i2;

              double Values[numTypes+1];

              for ( i1=1;i1<=numTypes;i1++) {
                     for ( i2=1;i2<=numTypes;i2++) {
                           if (i1<firstInClub ) {
                                  if (i2<firstInClub ) {
                                         double numChances= numOfOneType;
                                         if ( i1==i2 ) numChances= numOfOneType-1;
                                         matchprobability[i1][i2]= numChances/(numTypesOutOfClub*numOfOneType-1);
                                  }
                                  else {
                                         matchprobability[i1][i2]= 0;
                                  }
                           }
```

```
            else {
                  if (i2<firstInClub ) {
                        matchprobability[i1][i2]= 0;
                  }
                  else {
                        double numChances= numOfOneType;
                        if ( i1==i2 ) numChances= numOfOneType-1;
                        matchprobability[i1][i2]= numChances/(numTypesInClub*numOfOneType-1);
                  }
            }
      }
      matchprobability[i1][0]= 0;
}

for ( i=1;i<=numTypes;i++) {
      distributionOfLeaver[i] = 1.0/numTypes; // all leave
}


for ( i=1;i<=numTypes;i++) {
      Values[i]=0;
}
double NewValues[numTypes+1];

double ValueOfBirth=0;
long counter = 500;
while (counter-- >0) {
      ValueOfBirth=0;
      for ( i=1;i<=numTypes;i++) {
            ValueOfBirth += distributionOfLeaver[i]*Values[i];
      }


      for ( i1=1;i1<=numTypes;i1++) {
            NewValues[i1]=-m_cost+matchprobability[i1][0]*Values[i1]*m_discountFactor;
            for ( i2=1;i2<=numTypes;i2++) {
                  NewValues[i1] += matchprobability[i1][i2]*m_valueMatrix[i1][i2];
            }
            NewValues[i1]+=(1-matchprobability[i1][0])*clubFactorOfQueue*ValueOfBirth;

            if ( i1>=firstInClub ) NewValues[i1] -= clubfee;
      }

      for ( i=1;i<=numTypes;i++) {
            Values[i]=NewValues[i];
      }
}
```

```
BOOL ExPostAcceptanceMatrix[numTypes+1][numTypes+1];
BOOL allOK=TRUE;
for ( i1=1;i1<=numTypes;i1++) {
       for ( i2=1;i2<=numTypes;i2++) {
              double valueAccept = -m_cost + m_valueMatrix[i1][i2] +clubFactorOfQueue*ValueOfBirth;
              double valueReject = -m_cost+m_discountFactor*Values[i1];
              ExPostAcceptanceMatrix[i1][i2]=valueAccept>=valueReject;

              if ( (( i1<firstInClub ) == ( i2<firstInClub )  )
                     && (ExPostAcceptanceMatrix[i1][i2]==0)){
                     allOK=FALSE; // not all in club accept each other
              }
       }
}

long highestNotInClub;
// check whether the highest not in club wants to enter the club
if ( allOK) {
       highestNotInClub = firstInClub-1;
       double valueOfClub= - clubfee;
       for (i=firstInClub; i<=numTypes; i++){
              if (ExPostAcceptanceMatrix[i][highestNotInClub] ) {
                     valueOfClub += (-m_cost+m_valueMatrix[i][highestNotInClub]
                            +clubFactorOfQueue*ValueOfBirth) /numTypesInClub;
              }
              else {
                     valueOfClub += (-m_cost+m_discountFactor*Values[highestNotInClub])/numTypesInClub;
              }
       }
       double valueOutOfClub =Values[highestNotInClub];

       if ( valueOfClub > valueOutOfClub ) {
              allOK = FALSE;
       }
}



if (allOK) {


       outString.Format("\nClub 1-%d,%d-%d",highestNotInClub, firstInClub, numTypes);
       outFile.WriteString(outString);
       outString.Format("\nFee %f", clubfee);
       outFile.WriteString(outString);
```

```
		outString.Format("\nRebirthValue\t%f",ValueOfBirth);
		outFile.WriteString(outString);

		outString.Format("\ndistribution of leavers");
		outFile.WriteString(outString);

		for ( i1=1;i1<=numTypes;i1++) {
			outString.Format("\t%f",(float)distributionOfLeaver[i1]);
			outFile.WriteString(outString);
		}

		outString.Format("\nValues");
		outFile.WriteString(outString);

		for ( i1=1;i1<=numTypes;i1++) {
			outString.Format("\t%f",(float)Values[i1]);
			outFile.WriteString(outString);
		}


		outString.Format("\nfactor of queue");
		outFile.WriteString(outString);

		for ( i1=1;i1<=numTypes;i1++) {
			outString.Format("\t%f",(float)clubFactorOfQueue);
			outFile.WriteString(outString);
		}


		outString.Format("\nmatching probability");
		for ( i1=1;i1<=numTypes;i1++) {
			outString.Format("\n%d",i1);
			outFile.WriteString(outString);
			for ( i2=1;i2<=numTypes;i2++) {
				outString.Format("\t%f",(float)matchprobability[i1][i2]);
				outFile.WriteString(outString);
			}
		}


	}

}
```

```
outFile.Close();
```