ASSIGNMENT NO:2 **DATE:21/07/2015**

PROGRAM TITLE:Write a Program to implement the Havel-Hakimi Algorithm for Testing the existence of a Graph

THEORY: The Havel Hakimi algorithm gives a systematic approach to answer the question of
determining whether it is possible to construct a simple graph from a given degree
sequence.

Take as input a degree sequence S and determine if that sequence is graphical
That is, can we produce a graph with that degree sequence?

PROGRAM ALGORITHM:

```
Algo_havel-hakimi(S)//S is the degree sequence
{
      if(any degree greater than number of vertices)
      {
            Graph not possible;
      }
      if(odd number of odd degrees)
      {
            Graph not possible;
      }
      while(atleast one degree is non negative)
      {
            Reorder degree sequence in non-increasing order;
            Let k = first element in degree sequence;
            Remove first element in degree sequence;
            Subtract 1 from the next k terms;
      }
      if(the final degree sequence is all zero)
            Graph is Possible;
      else
            Graph is not possible;
}
```

PROGRAM CODE:

```
/*C++ Program to implement the Havel-Hakimi Algorithm for Testing the
existence of a Graph.*/
#include<iostream>
using namespace std;

/*Class Degseq and its associated functions*/
class Degseq
{
      int l;
      public:
            int *seq;
            Degseq(int);
            ~Degseq() ;
```

```
            void decsort(int);
            int checkzero(int);
            void display(int m)
            {
                  for(int i=m;i<l;i++)
                  {
                        cout<<"\t"<<seq[i];
                  }
                  cout<<endl;
            }
};

/*Parameterised Constructor*/
Degseq::Degseq(int i)
{
      l=i;
      seq=new int[l];
}

/*Destructor*/
Degseq::~Degseq()
{
      delete []seq;
}

/*Function to sort the remaining degree sequence*/
void Degseq::decsort(int m)
{
      int i,j,key;
      for(i=m+1;i<l;i++)
      {
            key=seq[i];
            for(j=i-1;(j>=0)&&(seq[j]<key);j--)
            {
                  seq[j+1]=seq[j];
            }
            seq[j+1]=key;//inserts the key at proper position
      }
      display(m);
}

/*Returns -1 if negative degree exists, 0 if positive degree exists or 1
if all elements are zero*/
int Degseq::checkzero(int m)
{
      int i;
      for(i=m;i<l;i++)
      {
            if(seq[i]<0)
                  return -1;
            else if(seq[i]>0)
                  return 0;
            else
                  continue;
      }
```

```
	return 1;//all are zero
}
int main()
{
	int l,i,c=0,k;
	cout<<"\n\tEnter the number of vertices::";
	cin>>l;
	Degseq ob(l);
	cout<<"\n\tEnter non-negative degree sequence.\n\t";

	/*Checks if any degree is greater than the number of vertices*/
	for(i=0;i<l;i++)
	{
		cin>>ob.seq[i];
		if(ob.seq[i]%2)
			c++;
		if(ob.seq[i]>=l)
		{
			cout<<"\n\tGraph Not Possible: Degree greater than equal
to number of vertices."<<endl;
			return 1;
		}
	}

	/*Checks for even number of odd degree vertices*/
	if(c%2)
	{
		cout<<"\n\tGraph Not Possible: Odd number of odd degrees
present."<<endl;
		return 1;
	}
	c=0;

	/*Performs the Havel-Hakimi according to the algorithm*/
	while(ob.checkzero(c)==0)
	{

		/*Reorder degree sequence in non-increasing order*/
		ob.decsort(c);

		/*Let k = first element in degree sequence*/
		k=ob.seq[c];
		c++;

		/*Subtract 1 from the next k terms*/
		for(i=c;k>0;ob.seq[i]--,i++,k--);

	}

	/*Final step to Check the existence of the graph*/
	if(ob.checkzero(c)==-1)
		cout<<"\n\tGraph Not Possible: Degree is negative."<<endl;
	else
		cout<<"\n\tGraph is Possible."<<endl;
	return 0;
```

```
}
```

OUTPUT:

Set 1:

```
Enter the number of vertices::5

Enter non-negative degree sequence.
4 3 3 3 1
4     3     3     3     1
2     2     2     0
1     1     0

Graph is Possible.
```

Set 2:

```
Enter the number of vertices::8

Enter non-negative degree sequence.
7 5 5 4 4 4 4 3
7     5     5     4     4     4     4     3
4     4     3     3     3     3     2
3     3     2     2     2     2
2     2     2     1     1
1     1     1     1
1     1     0

Graph is Possible.
```

Set 3:

```
Enter the number of vertices::8

Enter non-negative degree sequence.
7 7 6 5 4 3 2 2
7     7     6     5     4     3     2     2
6     5     4     3     2     1     1
4     3     2     1     0     0
2     1     0     0     -1

Graph Not Possible: Degree is negative.
```

Set 4:

```
Enter the number of vertices::5

Enter non-negative degree sequence.
1 2 1 1 2

Graph Not Possible: Odd number of odd degrees present.
```

DISCUSSION:

We are using parameterised constructor for creating the object so that no degree sequence of size 0 exists(ambiguity) and a Destructor to delete the dynamically allocated elements.

Instead of deleting the first degree from the degree sequence and adding an extra step of rearranging the array, we are not working with that part of the array in the subsequent steps.

While declaring the objects of the class within the switch case, we faced a problem that the scope of the object was not clearly known. To overcome this problem we put all the statements of that particular case within a scope.