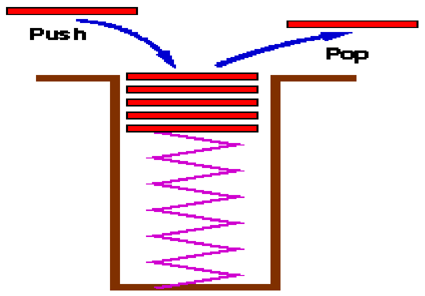
UNESCO-NIGERIA TECHNICAL & VOCATIONAL EDUCATION REVITALISATION PROJECT-PHASE II

NATIONAL DIPLOMA IN COMPUTER SCIENCE

DATA STRUCTURES AND ALGORITHMS



YEAR I- SEMESTER 2

THEORY

Table of Contents

WEEK 1

This week specific learning outcome is that the students should be able to use data attribute fields, sub fields, records and file. By this reason, the following activities are carried out to realize this objective:

To explain the following terms and using relevant examples:

a.) Data Type.

Data type of a variable is the set of values that the variable may assume.

- Basic data types in C : **int char float double**

- Basic data types in Pascal: **integer real char boolean**

Primitive Data Types

A new, primitive type is definable by enumerating the distinct values belonging to it. Such a type is called an *enumeration type*. Its definition has the form
TYPE T = (c1, c2, ... , cn)
T is the new type identifier, and the ci are the new constant identifiers.
Examples
TYPE shape = (rectangle, square, ellipse, circle)
TYPE color = (red, yellow, green)
TYPE sex = (male, female)
TYPE weekday = (Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday, Sunday)
TYPE currency = (franc, mark, pound, dollar, shilling, lira, guilder,
krone, ruble, cruzeiro, yen)
TYPE destination = (hell, purgatory, heaven)
TYPE vehicle = (train, bus, automobile, boat, airplane)
TYPE rank = (private, corporal, sergeant, lieutenant, captain, major, colonel, general)
TYPE object = (constant, type, variable, procedure, module)
TYPE structure = (array, record, set, sequence)
TYPE condition = (manual, unloaded, parity, skew)
The definition of such types introduces not only a new type identifier, but at the same time the set ofidentifiers denoting the values of the new type. These identifiers may then be used as constants throughout
the program, and they enhance its understandability considerably. If, as an example, we introduce variables
s, d, r, and b.
VAR s: sex
VAR d: weekday
VAR r: rank

then the following assignment statements are possible:
s := male
d := Sunday
r := major
b := TRUE
Evidently, they are considerably more informative than their counterparts
s := 1 d := 7 r := 6 b := 2 which are based on the assumption that c, d, r, and b are defined as integers and that the constants are mapped onto the natural numbers in the order of their enumeration.

b.) File

A **file** is a collection of logically related records stored on electric devices; e.g

students file, stock file, employee file etc.

ATTRIBUTES	NAME	AGE	SEX	MATRIC NO
VALUES	Paul	21	Male	800654

c.) **Record:**

A **record** is a collection of logically related data fields; e. g Data relating to students in students file. In a database table records are usually in rows. Therefore, the table below has three (3) records.

d.) Field:

A **field** is consecutive storage position of values. It is a unit of data within a record e. g student's number, Name, Age. In a database concept fields are usually in columns of a given table.

e.) Data item

Data items for example , date are called group items if they can be divided into subsystems.

f.)sub field

The date for instance is represented by the day, the month andumber is called an elementary item, because it can not be sub-divided into sud-items otherwise known as **sub fields** called . It is indeed treated as a single item.

g.) Character

is the smallest unit of information. It includes letters, digits and special symbols such as + (Plus sign), _(minus sign), \, /, $,a,b,…z, A,B,…Z etc. Every character requires one **byte** of memory unit for storage in computer system.

h.)Value range:

All possible values that could be assigned to a given atttibute of an entity set is called the range of values of the attribute.

WEEK 2

This week specific learning outcome is that the students should be able to use symbols, relations and graphs. By this reason, the following activities are carried out to realize this objective:

Examining the Different Classes of Edges

Graphs, in their simplest terms, are a collection of nodes and edges, but there are different kinds of edges:

1. Directed versus undirected edges
2. Weighted versus unweighted edges

When talking about using graphs to model a problem, it is usually important to indicate what class of graph you are working with.

Figure 1 shows three examples of graphs. Notice that graphs, unlike trees, can have sets of nodes that are disconnected from other sets of nodes. For example, graph (a) has two distinct, unconnected set of nodes. Graphs can also contain cycles. Graph (b) has several cycles. One such is the path from v_1 to v_2 to v_4 and back to v_1. Another one is from v_1 to v_2 to v_3 to v_5 to v_4 and back to v_1. (There are also cycles in graph (a).) Graph (c) does not have any cycles, as one less edge than it does number of nodes, and all nodes are reachable. Therefore, it is a tree.

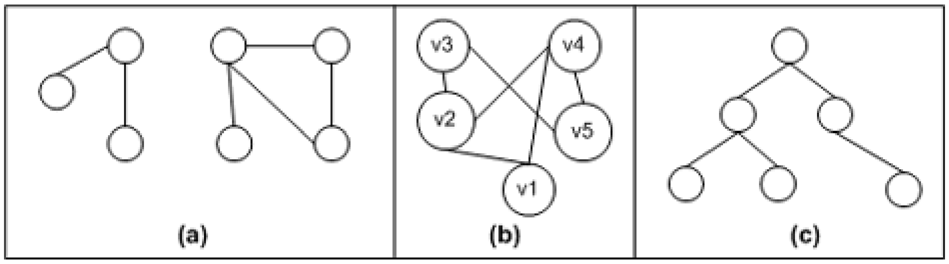


Figure 1. Three examples of graphs

Directed and Undirected Edges

The edges of a graph provide the connections between one node and another. By default, an edge is assumed to be bidirectional. That is, if there exists an edge between nodes *v* and *u*, it is assumed that one can travel from *v* to *u* and from *u* to *v*. Graphs with bidirectional edges are said to be *undirected graphs*, because there is no implicit direction in their edges.

For some problems, though, an edge might infer a one-way connection from one node to another. For example, when modeling the Internet as a graph, a hyperlink from Web page *v* linking to Web page *u* would imply that the edge between *v* to *u* would be unidirectional. That is, that one could navigate from *v* to *u*, but not from *u* to *v*. Graphs that use unidirectional edges are said to be *directed graphs*.

When drawing a graph, bidirectional edges are drawn as a straight line, as shown in Figure 1. Unidirectional edges are drawn as an arrow, showing the direction of the edge. Figure 2 shows a directed graph where the nodes are Web pages for a particular Web site and a directed edge from *u* to *v* indicates that there is a hyperlink from Web page *u* to Web page *v*. Notice that both *u* links to *v* and *v* links to *u*, two arrows are used—one from *v* to *u* and another from *u* to *v*.

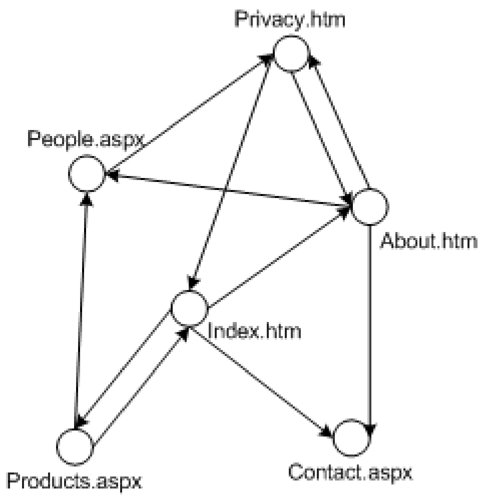


Figure 2. Model of pages making up a website

Weighted and Unweighted Edges

Typically graphs are used to model a collection of "things" and their relationship among these "things." For example, the graph in Figure 2 modeled the pages in a Web site and their hyperlinks. Sometimes, though, it is important to associate some cost with the connection from one node to another.

A map can be easily modeled as a graph, with the cities as nodes and the roads connecting the cities as edges. If we wanted to determine the shortest distance and route from one city to another, we first need to assign a cost from traveling from one city to another. The logical solution would be to give each edge a *weight*, such as how many miles it is from one city to another.

Figure 3 shows a graph that represents several cities in southern California. The cost of any particular path from one city to another is the sum of the costs of the edges along the path. The shortest path, then, would be the path with the least cost. In Figure 3, for example, a trip from San Diego to Santa Barbara is 210 miles if driving through Riverside, then to Barstow, and then back to Santa Barbara. The shortest trip, however, is to drive 100 miles to Los Angeles, and then another 30 up to Santa Barabara.

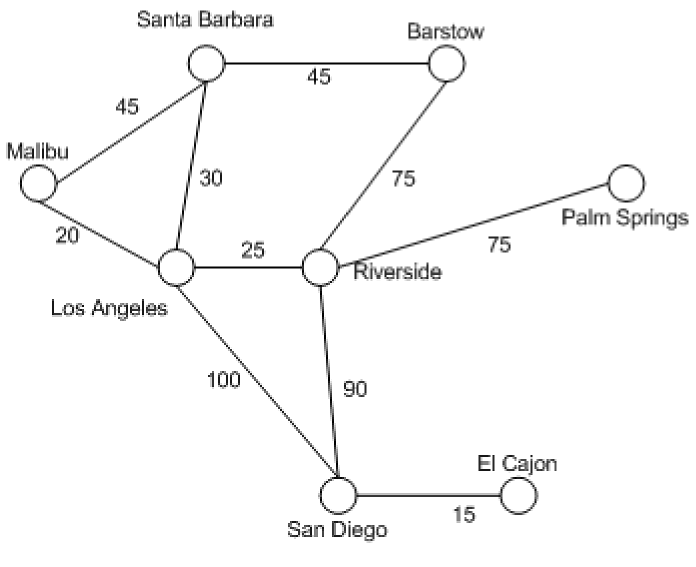


Figure 3. Graph of California cities with edges valued as miles

Realize that directionality and weightedness of edges are orthogonal. That is, a graph can have one of four arrangements of edges:

- Directed, weighted edges

- Directed, unweighted edges
- Undirected, weighted edges
- Undirected, unweighted edges

The graph's in Figure 1 had undirected, unweighted edges. Figure 2 had directed, unweighted edges, and Figure 3 used undirected, weighted edges.

WEE K 4

This week specific learning outcome is that the students should be able to write simple programs to carry out set operations. By this reason, the following activities a of a computer or other machine.

Once again, it is important to understand the following data types :

Recall that primitive (Also called built in) eg Integers, real, pointers, booleanY They are native or local to the language. The language knows their structure. They are part of the original design .

User-defined types: the language designer provides facilities for the user or programmer to defines his own data types. It is also called type constructor. Eg arrays in Fortran is the type constructor. Similarly, in Pascal we have: Type small Array = array 1:10 of integer

This Aarray Ais a type constructor and is used to generate a new type called small array.

As a result of the user-defined type above, we can have:

VAR X: small array: Just as we can have

VAR I: integer (i.e I is of type integer)

Here, both integer and small array are data types like X and I and are both variables.

An array is type constructor which helps to construct array types. The 2 popular types constructors are:

1) Arrays

2) Records

```
e.g   Type Person = Record
                    name         :string
                    age          :integer
                    occupation   : string
               M/status    (single/married)
```

Person here is a data type of record type and we can have: VAR X,Y,Z :person.

Values can also be assigned to X as: x:= Person(AJohn@,10,@students|@,single).

Other type constructors are:

a) Enumeration

Sets d) Sequence e) Relation Y

Problem: Find the greatest common divisor (GCD) of two integers, m and n.

Euclid's Algorithm:

```
while m is greater than zero:
  If n is greater than m, swap m and n.
  Subtract n from m.
n is the GCD
```

Program (in C):

```
int gcd(int m, int n)
/* precondition: m>0 and n>0.  Let g=gcd(m,n). */
 { while( m > 0 )
    { /* invariant: gcd(m,n)=g */
     if( n > m )
       { int t = m; m = n; n = t; } /* swap */
      /* m >= n > 0 */
      m -= n;
    }
  return n;
 }
```

WEEK 5

This week specific learning outcome is that the students should be able to solve problems requiring the application of string length, assignment, selection, insertion.

-Finding Concatenation of Strings

Let S1 and S2 be strings, the string consisting of the characters of S1 followed by the characters of S2 is called concatenation of S1 and S1.
This is donated by S1//S2. E.g. “STARLETS”//”//”DEFEAT”//”EA..GLET”
A string Y is called a substring of a string S if there exist strings X and Z such that S=X//Y//Z.

-finding Length of a string

The general form is LENGTH(string) and this will return the number of character(s) in a giving string.

i.) LENGTH(‘student’)=7
ii.) LENGTH(‘ ‘)=0
iii.) LENGTH(T)-(K+L-1)=LENGTH(T)-K-L+1

-finding Insertion

Suppose we want to insert a strings in a given text T so that S starts in position K. We denote this operation as INSERT(text, position, string). Eg.

INSERT(‘ABCDEFG’, 3,’XYZ’)=’ABXYZCDEFG’

-finding deletion

The general form is DELETE(text,position,length).eg.

i.) DELETE(‘ABCDEFG’,4,2)=’ABCFG’

ii.) DELETE(‘ABCDEFG’,0,2)=’ABCDEFG’

iii.) DELETE(T,K,L)=SUBSTRING(T,1,K-1)//SUBSTRING(T,K+L,LENTGH(T)-K-L+1)

iv.) DELETE(T,0,L)=T

v.) DELETE(T,INDEX(T,P),LENGTH(P))=DELETE('ABCDEFG',INDEX('ABCD EFG','CD') ,2)='ABEFG'

Deletion Algorithm:

A text T and a pattern P are in computer memory. This algorithm deletes every occurrence of P in T.

```
i.)  Find index of P in T. Set K=INDEX(T,P)
ii.) Repeat while K not equal to 0
        a. [Delete P from T.]
           Set T:= DELETE(T,INDEX(T,P),LENGTH(P))

        b. [Update index]
           Set K:= INDEX(T,P)
         [End of loop]
iii.)Write: T.
iv.) Exit.
```

REPLACEMENT

Suppose in a given T we want to replace the first occurrence of a patter P1 by a pattern P2, we will denote this operation by REPLACE(test,pattern1,pattern2)

REPLACE('XABYABZ,'AB','C')='XCYABZ'

REPLACE('XABYABZ,'BA','C')='XABYABZ'

This could also be done using the following algorithm:

Replacement Algorithm:

```
1. [Find index of P] Set K :=INDEX(T,P)
2. Repeat while K>0:
     a.) [Replace P by Q] Set T:=REPLACE(T,P,Q)
     b.) [Updarte] Set K:= INDEX(T,P)
   [Eend loop]
3.   Write: T
4. Exit.
```

Exercise

a.) T=XABYABZ
    P=AB
    Q=C
    REPLACE(T,P,Q)

b.) If T=XAB
    P=A
    Q=AB
    REPLACE(T,P,Q).

WEEK 6

This week specific learning outcome is that the students should be able to add a new item to the list.

Addition of a new item to the list

The pseudocode for the algorithm to add a new name to the list is as follows: Here are the steps:

```
Begin procedure
    Node [next free].name=new name
    P=start
    Follow pointers until node[p].pointer points to a name > new name
    Temp=next free
    Temp = next free
    Next free=node[]next free].pointer
    Node [temp]. pointer = node [p].pointer
    Node [p] . pointer = temp
End procedure
```

WEEK 7

This week specific learning outcome is that the students should be able to write the procedure for deleting an item from a linked list .

Deletion of an item from a list

```
Begin procedure

        If  start = 0  then  write ‘list is  empty’ and  exit procedure.

        P=start

        If deletename = node[start].name  then

                Temp =  node[start].pointer

                Node[start].pointer = next free

                Next free  = start

                Start = temp

        Else

                While deletename< >  node [node].pointer].pointer

                        P =  [node[p].pointer

                Endwhile

                (node[p) now  points  to  the  node  to  be  deleted; adjust  the  pointers)

                Temp = node [p].pointer

                Node[p].pointer = node[temp].pointer

                Node[temp]. pointer = nextfree

                Next free  = temp

        endif
```

WEEK 8

This week specific learning outcome is that the students should be able to write program sample to show basic graph class.

Basic graph class

The following code shows a basic graph class. The HashMap and LinkedList classes are the ones you have used in previous chapters. Alternatively, you could use the equivalent Java Collections Framework classes.

```
/**

* class Graph
*
*/
public class Graph{

protected HashMap adjacencyMap;

/**

* Initialize this Graph object to be empty.
*/
public Graph()
{
adjacencyMap = new HashMap();
}


/**

* Determines if this Graph contains no vertices.
*
* @return true - if this Graph contains no vertices.
*/
public boolean isEmpty()
{


return adjacencyMap.isEmpty();
}


/**

* Determines the number of vertices in this Graph.
*
* @return the number of vertices.
```

```
*/
public int size()
{


return adjacencyMap.size();
}


/**

* Returns the number of edges in this Graph object.
*
* @return the number of edges.
*/
public int getEdgeCount()
{

int count = 0;
for (int i=0;i<adjacencyMap.CAPACITY;i++){
if (adjacencyMap.keys[i] != null){
LinkedList edges = (LinkedList)
adjacencyMap.get(adjacencyMap.keys[i]);
count += edges.size();

}
}
return count;


}

/**

* Adds a specified object as a vertex
*
* @param vertex - the specified object
* @return true - if object was added by this call
*/
public boolean addVertex (Object vertex)
{

if (adjacencyMap.containsKey(vertex))

return false;
adjacencyMap.put (vertex, new LinkedList());
return true;

}

/**

* Adds an edge, and vertices if not already present*
* @param v1 - the beginning vertex object of the edge
* @param v2 - the ending vertex object of the edge
* @return true - if the edge was added by this call
*/
public boolean addEdge (Object v1, Object v2)
{


addVertex (v1); addVertex (v2);
LinkedList l = (LinkedList)adjacencyMap.get(v1);
l.add(v2);
return true;


}
}
```

WEEK 9-10

This week specific learning outcome is that the students should be able to implement sort algorithm with some features of linked lists to solve problems .

Sort Algorithm with some features of linked lists

```
/******************************************************************

-> This program is to sort the given integers
   in ascending order using address calculation sort

-> bins are maintained using linked lists

-> With the hash function used in the program it is
   only possible to sort integers that are < 100

-> This program works in microsoft vc++ 6.0 environment.

*******************************************************************/

#include<iostream.h>

class linkedlist
{
private:
 int n;
 linkedlist *next;
public:
 linkedlist* insert(int,linkedlist*);
 void display(linkedlist*);
 friend class sorting;
};

linkedlist* linkedlist::insert(int x,linkedlist*a)
{
 linkedlist *NEW;
 NEW=new linkedlist;
 NEW->n=x;
 NEW->next=NULL;

 if(a==NULL)
  a=NEW;
 else          // search for the correct position to insert
 {
  linkedlist *l;
  l=a;
  if(x<l->n)
  {
   NEW->next=l;
   a=NEW;
  }
  else
  {
   while( l->n<x && l->next!=NULL )
    l=l->next;
   l->next=NEW;
  }
 }
 return a;
}

void linkedlist::display(linkedlist*a)
{
 while(a!=NULL)
 {
  cout<<a->n<<’\t’;
  a=a->next;
 }
 cout<<”NULL\n”;
}

class sorting
{
private:
 int n;
 linkedlist *array;
 linkedlist *bin[6];
public:
 void input();
 void add_calc_sort();
 void output();
};

void sorting::input()
{
 cout<<"**********************************************************\n";
 cout<<"This program sorts the given integers in ascending order\n"
  <<"   using address calculation sort algorithm \n";
 cout<<"**********************************************************\n";

 cout<<"Enter how many numbers you are going to enter ::";
 cin>>n;

 array=NULL;
 cout<<"Now enter your numbers only in the range (0-99) ::\n";
 for(int i=1;i<=n;i++)
 {
  int x;
  cin>>x;

  linkedlist *l;

  linkedlist *NEW;
  NEW=new linkedlist;
  NEW->n=x;
  NEW->next=NULL;

  if(array==NULL)
   array=NEW;
  else
   l->next=NEW;
  l=NEW;
 }
}

void sorting::add_calc_sort()
{
 //Hash the numbers in to the five bins

 linkedlist obj;
 int i;
 for(i=1;i<=5;i++)
  bin[i]=NULL;

 while(array!=NULL)
 {
  if( array->n >=0 && array->n <=19)
   bin[1]=obj.insert(array->n,bin[1] );
  else if( array->n >=20 && array->n <=39)
   bin[2]=obj.insert(array->n,bin[2] );
  else if( array->n >=40 && array->n <=59)
   bin[3]=obj.insert(array->n,bin[3] );
  else if( array->n >=60 && array->n <=79)
   bin[4]=obj.insert(array->n,bin[4] );
  else if( array->n >=80 && array->n <=99)
   bin[5]=obj.insert(array->n,bin[5] );

  array=array->next;
 }

 cout<<"\nThe contents of the bins are ::\n";
 for(i=1;i<=5;i++)
 {
  cout<<" ( "<<i<<" ) ::";
  obj.display(bin[i]);
 }

 //collect from all the bins
 array=NULL;
 for(i=1;i<=5;i++)
 {
  linkedlist *l;
  while(bin[i]!=NULL)
  {
   linkedlist *NEW;
   NEW=new linkedlist;

   NEW->n=bin[i]->n;
   NEW->next=NULL;

   if(array==NULL)
    array=NEW;
   else
    l->next=NEW;
   l=NEW;

   bin[i]=bin[i]->next;
  }
 }
}

void sorting::output()
{
 cout<<”After sorting the elements are ::\n”;
 linkedlist obj;
 obj.display(array);
}

int main()
{
 sorting obj;
 obj.input();
 obj.add_calc_sort();
 obj.output();
 return 0;
```

```
}
/******************************************************************

SAMPLE OUTPUT ::

**********************************************************
This program sorts the given integers in ascending order
   using address calculation sort algorithm
**********************************************************
Enter how many numbers you are going to enter ::7
Now enter your numbers only in the range (0-99) ::
70
65
60
55
50
45
40

The contents of the bins are ::
 ( 1 ) ::NULL
 ( 2 ) ::NULL
 ( 3 ) ::40     45      50      55      NULL
 ( 4 ) ::60     65      70      NULL
 ( 5 ) ::NULL
After sorting the elements are ::
40      45      50      55      60      65      70      NULL
Press any key to continue
```

WEEK 11

This week specific learning outcome is that the students should be able to implement selection sorting technique to arrange the following set of data:

1.1, 1.2, 1.3, 1.4, 1.5, 1.6 , and 1.7

Selection Sort

```
/**************************************************************

-> This C++ program is to perform selection sort.

-> This program works in microsoft vc++ 6.0 environment.

-> The numbers are sorted in increasing order.

**************************************************************/

#include<iostream.h>

class sorting
{
private:
 int n;
 double * array;
public:
 void input();
 void selectionsort();
 void output();
};

void sorting::input()
{
 cout<<"*******************************************************\n"
  <<"This program sorts numbers in increasing order"
  <<"\n\t\tusing selection sort technique\n"
  <<"*****************************************************\n";

 cout<<"Enter how many numbers you are going to enter for sorting ::";
 cin>>n;
 array=new double[n+1];
 cout<<"Now enter your elements ::\n";
 for(int i=1;i<=n;i++)
  cin>>array[i];
}

void sorting::selectionsort()
{
 for(int i=1;i<=n;i++)
 {
  double min=array[i];
  int min_p=i;
        for(int j=i+1;j<=n;j++)
   if(array[j]<min)
    min=array[j],min_p=j;
  if(i!=min_p)
  {
   double t=array[i];
   array[i]=min;
   array[min_p]=t;
  }
  }
}

void sorting::output()
{
 cout<<"Now the sorted numbers are ::\n";
 for(int i=1;i<=n;i++)
  cout<<array[i]<<'\t';
 cout<<endl;
}
```

```
int main()
{
 sorting obj;
 obj.input();
 obj.selectionsort();
 obj.output();
 return 0;
}

/*************************************************************************

SAMPLE OUTPUT ::

******************************************************
This program sorts numbers in increasing order
        using selection sort technique
******************************************************
Enter how many numbers you are going to enter for sorting ::7
Now enter your elements ::
1.7
1.6
1.5
1.4
1.3
1.2
1.1
Now the sorted numbers are ::
1.1     1.2     1.3     1.4     1.5     1.6     1.7
Press any key to continue

*************************************************************************/
```

WEEK 12-13

This week specific learning outcome is that the students should be able to implement insertion sorting algorithm for the following set of data:

Insertion Sort

```
/**********************************************************


-> This C++ program is to perform insertion sort.


-> This program works in microsoft vc++ 6.0 environment.


-> The numbers are sorted in increasing order.


**********************************************************/


#include<iostream.h>


class sorting
{
private:
 int n;
 double *array;
public:
 void input();
 void insertionsort();
 void output();
};


void sorting::input()
{
 cout<<"*****************************************************\n"
  <<"This program sorts numbers in increasing order"
  <<"\n\t\tusing insertion sort technique\n"
  <<"*****************************************************\n";
 cout<<”Enter how many numbers you are going to enter for sorting ::”;
 cin>>n;
 array=new double[n+1];
 cout<<”Now enter your elements ::\n”;
 for(int i=1;i<=n;i++)
  cin>>array[i];
}

void sorting::insertionsort()
{
 for(int i=1;i<=n;i++)
 {
  double x=array[i];
  for(int j=i-1;j>0&&x<array[j];j–)
   array[j+1]=array[j];
  array[j+1]=x;
 }
}
void sorting::output()
{
 cout<<”Now the sorted numbers are ::\n”;
 for(int i=1;i<=n;i++)
  cout<<array[i]<<’\t’;
 cout<<endl;
}

int main()
{
 sorting obj;
 obj.input();
 obj.insertionsort();
 obj.output();
 return 0;
}

/*******************************************************************
```

SAMPLE OUTPUT ::

```
*****************************************************
This program sorts numbers in increasing order
                using insertion sort technique
*****************************************************
Enter how many numbers you are going to enter for sorting ::7
Now enter your elements ::
1.7
1.6
1.5
1.4
1.3
1.2
1.1
 Now the sorted numbers are ::
1.1     1.2     1.3     1.4     1.5     1.6     1.7
Press any key to continue

**********************************************************************/
```

Week 14-15

This week specific learning outcome is that the students should be able to implement Rank sort algorithm to arrange some given data in increasing order.

Rank Sort

```
/*********************************************************

-> This C++ program is to perform rank sort.

-> This program works in microsoft vc++ 6.0 environment.

-> The numbers are sorted in increasing order.

*********************************************************/

#include<iostream.h>

class sorting
{
private:
 int n,rank[50];
 double *array;
public:
 void input();
 void ranksort();
 void output();
};

void sorting::input()
{
 cout<<"*****************************************************\n"
  <<"This program sorts numbers in increasing order"
  <<"\n\t\tusing rank sort technique\n"
  <<"*****************************************************\n";
 cout<<”Enter how many numbers you are going to enter for sorting ::”;
 cin>>n;
 array=new double[n+1];
 cout<<”Now enter your elements ::\n”;
 for(int i=1;i<=n;i++)
  cin>>array[i];
}

void sorting::ranksort()
{
  int i,j;
  double b[50];
        for(i=1;i<=n;i++)
                rank[i]=0;
        for(i=1;i<=n;i++)
                for(j=1;j<=i;j++)
                        if(array[j]>array[i])rank[j]++;
                        else rank[i]++;
        for(i=1;i<=n;i++)
                b[rank[i]]=array[i];
        for(i=1;i<=n;i++)
                array[i]=b[i];
}

void sorting::output()
{
 cout<<”Now the sorted numbers are ::\n”;
 for(int i=1;i<=n;i++)
  cout<<array[i]<<’\t’;
 cout<<endl;
}

int main()
{
 sorting obj;
 obj.input();
 obj.ranksort();
 obj.output();
 return 0;
}
/*********************************************************************

SAMPLE OUTPUT ::

*****************************************************
This program sorts numbers in increasing order
            using rank sort technique
*****************************************************
Enter how many numbers you are going to enter for sorting ::7
Now enter your elements ::
1.7
1.6
1.5
1.4
1.3
1.2
1.1
Now the sorted numbers are ::
1.1    1.2    1.3    1.4    1.5    1.6    1.7
Press any key to continue
*********************************************************************/
```