Subject : Data Structures
Semester : 3rd
Topic : Array 1 Dimensional

PROGRAM ON TRAVERSING A LINEAR ARRAY.

Algorithm 4.1: (Traversing a Linear Array) Here LA is a linear array with lower bound LB and upper bound UB. This algorithm traverses LA applying an operation PROCESS to each element of LA.

1. [Initialize counter.] Set K := LB.
2. Repeat Steps 3 and 4 while K ≤ UB.
3. [Visit element.] Apply PROCESS to LA[K].
4. [Increase counter.] Set K := K + 1.

   [End of Step 2 loop.]
5. Exit.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int LA[]={0,11,22,33,44,55,66,77,88,99,100};
int K,LB,UB;
clrscr();
LB=1;
  UB=10;
K=LB;
while(K<=UB)
{
     printf("%d\n",LA[K]);
     K=K+1;
}
getch();
}
```

Algorithm 4.1′: (Traversing a Linear Array) This algorithm traverses a linear array LA with lower bound LB and upper bound UB.

1. Repeat for K = LB to UB:
   Apply PROCESS to LA[K].

   [End of loop.]
2. Exit.

PROGRAM ON INSERTING ELEMENT INTO LINEAR ARRAY.

Algorithm 4.2: (Inserting into a Linear Array) INSERT (LA, N, K, ITEM)
Here LA is a linear array with N elements and K is a positive integer such that K≤ N. This algorithm inserts an element ITEM into the Kth position in LA.

1. [Initialize counter.] Set J : = N.
2. Repeat Steps 3 and 4 while J ≥ K.
3. [Move Jth element downward.] Set LA[J + 1] := LA[J].
4. [Decrease counter.] Set J := J – 1.
   [End of Step 2 loop.]
5. [Insert element.] Set LA[K] := ITEM.
6. [Reset N.] Set N := N + 1.
7. Exit.

```
#include<stdio.h>
#include<conio.h>
void INSERT(int LA[], int &N, int K, int ITEM)
{
int J;
J=N ;
while(J>=K)
{
     LA[J+1]=LA[J];
     J=J-1;
}
LA[K]=ITEM;
N=N+1;
}

void main()
{
int LA[20]={0,11,22,33,44,55,66,77,88,99,100};
int K,LB,UB;
clrscr();
LB=1;
UB=10;
INSERT(LA, UB, 6, 999);

K=LB;
while(K<=UB)
{
     printf("%d\n",LA[K]);
     K=K+1;
}
getch();}
```

PROGRAM ON DELETING ELEMENT FROM LINEAR ARRAY.

Algorithm 4.3: (Deleting from a Linear Array) DELETE(LA, N, K, ITEM)

Here LA is a linear array with N elements and K is a positive integer such that $K \leq N$. This algorithm deletes the Kth element from LA.

1. Set ITEM := LA[K].
2. Repeat for J = K to N – 1:
   [Move J + 1st element upward.] Set LA[J] := LA[J + 1].
   [End of loop.]
3. [Reset the number N of elements in LA.] Set N: = N – 1.
4. Exit.

```
#include<stdio.h>
#include<conio.h>
void DELETE(int LA[], int &N, int K, int ITEM)
{
int J;
ITEM=LA[K];
for(J=K;J<=N-1;J++)
{
     LA[J]=LA[J+1];
}
N=N-1;
}
void main()
{
int LA[20]={0,11,22,33,44,55,66,77,88,99,100};
int K,LB,UB;
clrscr();
LB=1;
UB=10;
DELETE(LA, UB, 6, 999);
K=LB;
while(K<=UB)
{
     printf("%d\n",LA[K]);
     K=K+1;
}
getch();
}
```

PROGRAM ON BUBBLE SORT.

Example 4.7

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

We apply the bubble sort to the array A. We discuss each pass separately.

Pass 1. We have the following comparisons:

(a) Compare A_1 and A_2. Since 32 < 51, the list is not altered.

(b) Compare A_2 and A_3. Since 51 > 27, interchange 51 and 27 as follows:

32, (27), (51), 85, 66, 23, 13, 57

(c) Compare A_3 and A_4. Since 51 < 85, the list is not altered.

(d) Compare A_4 and A_5. Since 85 > 66, interchange 85 and 86 as follows:

32, 27, 51, (66), (85), 23, 13, 57

(e) Compare A_5 and A_6. Since 85 > 23, interchange 85 and 23 as follows:

32, 27, 51, 66, (23), (85), 13, 57

(f) Compare A_6 and A_7. Since 85 > 13, interchange 85 and 13 to yield:

32, 27, 51, 66, 23, (13), (85), 57

(g) Compare A_7 and A_8. Since 85 > 57, interchange 85 and 51 to yield:

32, 27, 51, 66, 23, 13, (57), (85)

At the end of this first pass, the largest number, 85, has moved to the last position. However, the rest of the numbers are not sorted, even though some of them have changed their positions.

For the remainder of the passes, we show only the interchanges.

Pass 2. (27), (33), 51, 66, 23, 13, 57, 85

27, 33, 51, (23), (66), 13, 57, 85

27, 33, 51, 23, (13), (66), 57, 85

27, 33, 51, 23, 13, (57), (66), 85

At the end of Pass 2, the second largest number, 66, has moved its way down to the next-to-last position.

Pass 3. 27, 33, (23), (51), 13, 57, 66, 85

27, 33, 23, (13), (51), 57, 66, 85

Pass 4. 27, (23), (33), 13, 51, 57, 66, 85

27, 23, (13), (33), 51, 57, 66, 85

Pass 5. (23), (27), 13, 33, 51, 57, 66, 85

23, (13), (27), 33, 51, 57, 66, 85

Pass 6. (13), (23), 27, 33, 51, 57, 66, 85

Pass 6 actually has two comparisons, A_1 with A_2 and A_2 and A_3. The second comparison does not involve an interchange.

Pass 7. Finally, A_1 is compared with A_2. Since 13 < 23, no interchange takes place.

Since the list has 8 elements; it is sorted after the seventh pass. (Observe that in this example, the list was actually sorted after the sixth pass. This condition is discussed at the end of the section.)

Algorithm 4.4: (Bubble Sort) BUBBLE(DATA, N)
Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

1. Repeat Steps 2 and 3 for K = 1 to N – 1.
2. Set PTR := 1. [Initializes pass pointer PTR.]
3. Repeat while PTR ≤ N – K: [Executes pass.]
   (a) If DATA[PTR] > DATA[PTR + 1], then:
   Interchange DATA[PTR] and DATA[PTR + 1].
   [End of If structure.]
   (b) Set PTR := PTR + 1.
   [End of inner loop.]
   [End of Step 1 outer loop.]
4. Exit.

```
#include<stdio.h>
#include<conio.h>
void BUBBLE(int *DATA, int N)
{
int K,PTR,TEMP;
for(K=1;K<=N-1;K++)
{
      PTR=1;
      while(PTR <= N-K)
      {
            if(DATA[PTR] > DATA[PTR+1])
            {
                  TEMP=DATA[PTR];
                  DATA[PTR]=DATA[PTR+1];
                  DATA[PTR+1]=TEMP;
            }
      PTR=PTR+1;
      }
}
}
void main()
{
int LA[]={0,11,33,22,44,55,88,66,77,100,99};
int K,LB,UB;
clrscr();
LB=1;
UB=10;
BUBBLE(LA, UB);
K=LB;
while(K<=UB)
{
      printf("%d\n",LA[K]);
```

```
     K=K+1;
}
getch();
}
```

ALGORITHM AND PROGRAM ON LINEAR SEARCH.

Algorithm

LINEAR(DATA, N, ITEM, LOC)

Here DATA is a linear array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA, or sets LOC= 0 if the search is unsuccessful.

```
1. [Insert ITEM at the end of DATA.] Set DATA[N + 1] := ITEM.
2. [Initialize counter.] Set LOC= 1.
3. [Search for ITEM.]
     Repeat while DATA[LOC] != ITEM:
     Set LOC:= LOC+1.
     [End of loop.]
4. [Successful?] If LOC = N+1, then: Set LOC := 0.
5. Exit.
```

Program

```
#include<stdio.h>
#include<conio.h>
void LINEAR(int *DATA, int N, int ITEM, int LOC)
{
DATA[N+1]=ITEM;
LOC=1;
while(DATA[LOC] != ITEM)
{
     LOC=LOC+1;
}
if(LOC==N+1)
{
     LOC=0;
}
printf("%d",LOC);
}

void main()
{
int LA[20]={0,11,33,22,44,55,88,66,77,100,99};
int K,LB,UB;
clrscr();
LB=1;
```

```
UB=10;
LINEAR(LA, UB, 77, 1);
getch();
}
```

ALGORITHM AND PROGRAM ON BINARY SEARCH.

Algorithm

BINARY(DATA, LB, UB, ITEM, LOC)

Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID denote, respectively, the beginning, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

```
1. [Initialize segment variables.]
   Set BEG:= LB, END := UB and MID = INT(BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG <= END and DATA[MID] != ITEM.
3.      If ITEM < DATA[MID], then:
        Set END:= MID - 1.
        Else:
        Set BEG := MID + 1.
        [End of If structure.]
4.      Set MID:= INT(BEG + END)/2).
  [End of Step 2 loop.]
5. If DATA[MID = ITEM, then:
        Set LOC= MID.
   Else:
        Set LOC:= NULL.
  [End of If structure.]
6. Exit.
```

Program

```
#include<stdio.h>
#include<conio.h>
void BINARY(int DATA[], int LB, int UB, int ITEM, int LOC)
{
int BEG, END, MID;
BEG=LB;
END=UB;
MID=(BEG+END)/2;
while( BEG<=END && DATA[MID] != ITEM)
{
     if(ITEM<DATA[MID])
     {
          END=MID-1;
```

```
      }
      else
      {
            BEG=MID+1;
      }
MID=(BEG+END)/2;
}
if(DATA[MID]==ITEM)
{
      LOC=MID;
}
else
{
      LOC=-1;
}
printf("%d",LOC);
}
void main()
{
int LA[20]={0,11,22,33,44,55,66,77,88,99,100};
int K,LB,UB;
clrscr();
LB=1;
UB=10;
BINARY(LA, LB, UB, 77, 1);
getch();
}
```

Let DATA be the following sorted 13-element array:

DATA: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

We apply the binary search to DATA for different values of ITEM.

(a) Suppose ITEM = 40. The search for ITEM in the array DATA is pictured in Fig. 4.6, where the values of DATA[BEG] and DATA[END] in each stage of the

algorithm are indicated by circles and the value of DATA[MID] by a square. Specifically, BEG, END and MID will have the following successive values:

1. Initially, BEG = 1 and END = 13. Hence

   MID = INT[(1 + 13)/2] = 7 and so DATA[MID] = 55

2. Since 40 < 55, END has its value changed by END = MID − 1 = 6. Hence

   MID = INT[(1 + 6)/2] = 3 and so DATA[MID] = 30

3. Since 40 > 30, BEG has its value changed by BEG = MID + 1 = 4. Hence

   MID = INT[(4 + 6)/2] = 5 and so DATA[MID] = 40

We have found ITEM in location LOC = MID = 5.

(1) (11) 22, 30, 33, 40, 44, [55], 60, 66, 77, 80, 88, (99)
(2) (11) 22, [30], 33, 40, (44) 55, 60, 66, 77, 80, 88, 99
(3) 11, 22, 30, (33) [40] (44) 55, 60, 66, 77, 80, 88, 99 [Successful]

Fig. 4.6 *Binary Search for ITEM = 40*

(b) Suppose ITEM = 85. The binary search for ITEM is pictured in Fig. 4.7. Here BEG, END and MID will have the following successive values:

1. Again initially, BEG = 1, END = 13, MID = 7 and DATA[MID] = 55.
2. Since 85 > 55, BEG has its value changed by BEG = MID + 1 = 8. Hence

   MID = INT[(8 + 13)/2] = 10 and so DATA[MID] = 77

3. Since 85 > 77, BEG has its value changed by BEG = MID + 1 = 11. Hence

   MID = INT[(11 + 13)/2] = 12 and so DATA[MID] = 88

4. Since 85 < 88, END has its value changed by END = MID − 1 = 11. Hence

   MID = INT[(11 + 11)/2] = 11 and so DATA[MID] = 80

(Observe that now BEG = END = MID = 11.)

Since 85 > 80, BEG has its value changed by BEG = MID + 1 = 12. But now BEG > END. Hence ITEM does not belong to DATA.

(1) (11) 22, 30, 33, 40, 44, [55], 60, 66, 77, 80, 88, (99)
(2) 11, 22, 30, 33, 40, 44, 55, (60) 66, [77], 80, 88, (99)
(3) 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, (80) [88] (99)
(4) 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, (80) 88, 99 [Unsuccessful]

Fig. 4.7 *Binary Search for ITEM = 85*