**UCS 2312 Data Structures Lab**

**Exercise 1: Array ADT and its applications**

**Date of Exercise: 05.09.2023**

Create an ADT for the array data structure with the following functions. *arrADT* will have the integer array and size. [CO1, K3]

1. create(arrADT,size, array) – Create the array with the required number of elements
2. deleteAt(arrADT, pos ) – Delete the specified element
3. insertAtEvery(arrADT,data) – Insert data before every element
4. search(arrADT, key) – return the position of the second occurrence of the element. If found return the position, otherwise return 1
5. printArray(arrADT) – prints the elements of the array
6. findPeek(arrADT, int \*) – return a set of peek elements

Given an array **arr[]** of integers. Find a peak element i.e. an element that is **not smaller** than its neighbors.

**Note:** For corner elements, we need to consider only one neighbor.

**Example:**

**Input:** array[] = {10, 20, 15, 2, 23, 90, 67}  
**Output:** 20, 90  
**Explanation:**The element 20 has neighbors 10 and 15, both of them are less than 20, similarly 90 has neighbors 23 and 67.

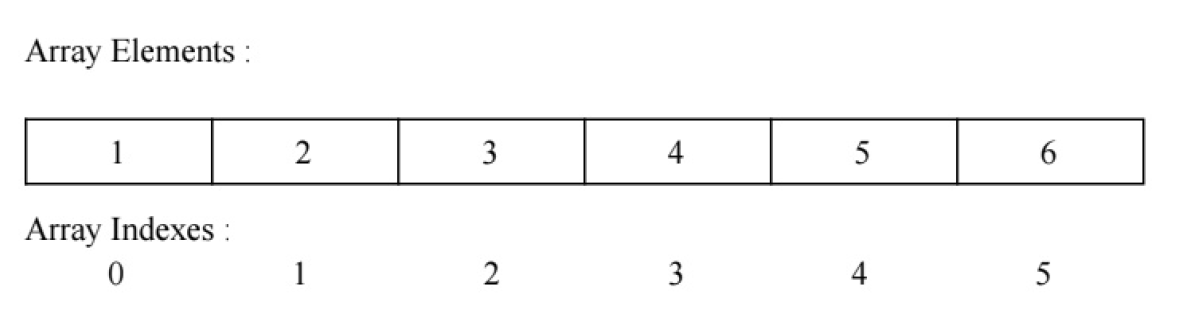
Write a program in C to test the operations of arrADT with the following test cases:

|  |  |
| --- | --- |
| **Operation** | **Expected Output** |
| create(arrADT,20,[2,4,6,8,10]) | 2,4,6,8,10 |
| deleteAt(arrADT, 3) | 2,4,6,10 |
| insertAtEvery(arrADT,1) | 1,2,1,4,1,6,1,10 |
| search(arrADT,1) | 2 |
| search(arrADT,2) | -1 |
| printArray(arrADT) | 1,2,1,4,1,6,1,10 |
| create(arrADT,20,[10,20,15,2,23,90,67]) | 20,90 |
| create(arrADT,20,[1,2,3,4,4]) | -1 |

Best practices to be followed:

* Design before coding
* Usage of algorithm notation
* Use of multi-file C program
* Versioning of code

**Data Structure – Array:**

****

**Algorithm –**

**Algorithm: Deleting an element from specified position**

Input – Pointer to array, position of element to be deleted

Output – void

1. for (i=pos-1; i<size-1; i++)

A->a[i] = A->a[i+1]

1. A->size-=1

**Algorithm: Insert data before every element**

Input – Pointer to array, data to be inserted

Output – void

1. for (i=A->size-1; i>=0; i--)

A->a[(i\*2) +1] = A->a[i]

A->a[i\*2] = data

1. A->size\*=2

**Algorithm: Return the position of the second occurrence of the element. If found return the position, otherwise return 1**

Input – Pointer to array, data to be found

Output – int

1. C=0 and pos=-1
2. for (i=0; i<A->size; i++)

if c==2

break

if A->a[i]==key

pos=i+1

c+=1

1. return pos

**Algorithm: Return a set of peek elements**

Input – Pointer to array, array to store peak elements

Output – int

1. c=0 and l=A->size
2. if A->a[0] > A->a[1]

p[c++]=A->a[0]

1. for (i=1; i<l-1; i++)

if A->a[i] > A->a[i-1] && A->a[i] > A->a[i+1]

p[c++]=A->a[i]

1. if A->a[l-1] > A->a[l-2]

p[c++]=A->a[l-1]

1. return c

**main.c code:**

#include<stdio.h>

#include<stdlib.h>

#include"arrADT.h"

void main()

{

struct arrADT \*A;

A=(struct arrADT \*)malloc(sizeof(struct arrADT));

int size;

printf("Enter the size of the array : ");

scanf("%d",&size);

int a[size],p[size];

printf("Enter %d array elements : \n",size);

for(int i=0;i<size;i++)

{

scanf("%d",&a[i]);

}

printf("\n");

create(A,size,a);

printArray(A);

printf("Enter the position of element to be deleted : ");

int pos;

scanf("%d",&pos);

deleteAt(A,pos);

printArray(A);

printf("Enter the data to be inserted : ");

int data;

scanf("%d",&data);

printf("Inserting At Front :\n");

insertAtFront(A,data);

printArray(A);

printf("Inserting At Middle :\n");

insertAtMiddle(A,data);

printArray(A);

printf("Inserting At End :\n");

insertAtEnd(A,data);

printArray(A);

printf("Deleting At Front :\n");

deleteAtFront(A);

printArray(A);

printf("Deleting At Middle :\n");

deleteAtMiddle(A);

printArray(A);

printf("Deleting At End :\n");

deleteAtEnd(A);

printArray(A);

printf("Inserting at every position :\n");

insertAtEvery(A,data);

printArray(A);

printf("Enter the key to search : ");

int key;

scanf("%d",&key);

printf("The position of %d is %d\n",key,search(A,key));

printf("Peek Values are : ");

int c=findPeek(A,p);

for(int i=0;i<c;i++)

{

printf("%d ",p[i]);

}

}

**arrADT.h code:**

struct arrADT

{

int size;

int a[100];

};

void create(struct arrADT \*A,int size,int array[])

{

A->size=size;

for(int i=0;i<size;i++)

{

A->a[i]=array[i];

}

}

void printArray(struct arrADT \*A)

{

for(int i=0;i<(A->size);i++)

{

printf("%d ",A->a[i]);

}

printf("\n");

}

void deleteAt(struct arrADT \*A,int pos)

{

for(int i=pos-1;i<((A->size)-1);i++)

{

A->a[i]=A->a[i+1];

}

A->size=(A->size)-1;

}

void deleteAtFront(struct arrADT \*A)

{

int pos=1;

for(int i=pos-1;i<((A->size)-1);i++)

{

A->a[i]=A->a[i+1];

}

A->size=(A->size)-1;

}

void deleteAtMiddle(struct arrADT \*A)

{

int pos=(A->size)/2;

pos+=1;

for(int i=pos-1;i<((A->size)-1);i++)

{

A->a[i]=A->a[i+1];

}

A->size=(A->size)-1;

}

void deleteAtEnd(struct arrADT \*A)

{

int pos=(A->size);

for(int i=pos-1;i<((A->size)-1);i++)

{

A->a[i]=A->a[i+1];

}

A->size=(A->size)-1;

}

void insertAtEvery(struct arrADT \*A,int data)

{

for(int i=(A->size)-1;i>=0;i--)

{

A->a[(i\*2)+1]=A->a[i];

A->a[i\*2]=data;

}

A->size=(A->size)\*2;

}

void insertAtFront(struct arrADT \*A,int data)

{

for(int i=(A->size)-1;i>=0;i--)

{

A->a[i+1]=A->a[i];

}

A->a[0]=data;

A->size=(A->size)+1;

}

void insertAtMiddle(struct arrADT \*A,int data)

{

for(int i=(A->size)-1;i>=((A->size)/2);i--)

{

A->a[i+1]=A->a[i];

}

A->a[((A->size)/2)]=data;

A->size=(A->size)+1;

}

void insertAtEnd(struct arrADT \*A,int data)

{

A->a[(A->size)]=data;

A->size=(A->size)+1;

}

int search(struct arrADT \*A,int key)

{

int c=0,pos=-1;

for(int i=0;i<(A->size);i++)

{

if(c==2)

break;

if(A->a[i]==key)

{

pos=i+1;

++c;

}

}

return pos;

}

int findPeek(struct arrADT \*A,int p[])

{

int c=0,l=A->size;

if(A->a[0]>A->a[1])

{

p[c++]=A->a[0];

}

for(int i=1;i<l-1;i++)

{

if(A->a[i]>A->a[i-1] && A->a[i]>A->a[i+1])

{

p[c++]=A->a[i];

}

}

if(A->a[l-1]>A->a[l-2])

{

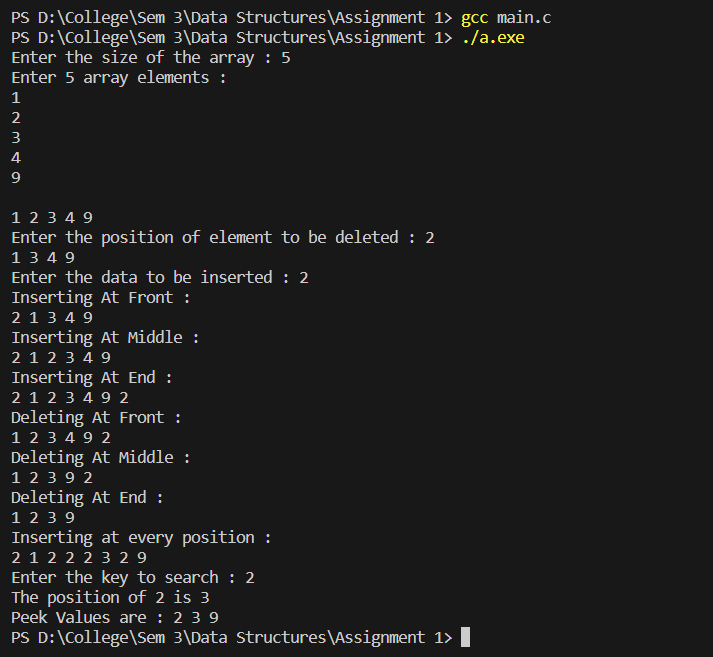
p[c++]=A->a[l-1];

}

return c;

}

**Output:**



**Learning Outcome:**

