**Data Structures and Algorithms ASSIGNMENT**

**SUBMITTED BY: AJAY KUMAR SINGH**

**ROLL NUMBER: 17CSE04**

1. Write a program in C language that can do the following tasks:

* reads a file
* breaks each line into words
* strips whitespace and punctuation from the words
* Converts them to lowercase.

Input: The File is attached below.

Output: As mentioned above.

**Code:**

#include <stdio.h> // for i/o

#include <ctype.h> // for string processing

*void* readbyword(FILE \*fp);

*void* removepunct(FILE \*fp);

*char* lower(*char* c);

*int* main(){

FILE \*fp; // file pointer

*int* choice;

fp = fopen("file.txt","r"); // opens the file.txt file , "r" specifies to open in read mode

printf(" \n 1. Print file word by Word\n 2. Strip whitespace and punctuation & print in lowercase\n\n Enter your choice : ") ;

scanf("%d",&choice);

switch(choice){

case 1: readbyword(fp);

break;

case 2: removepunct(fp);

break;

default : printf("enter correct !! bye");

}

fclose(fp); // closing of file

return 0;

}

*void* readbyword(FILE \*fp){ // function to read file and output each word

*char* c;

while((c = fgetc(fp)) != EOF){

if(c == ' ' || c == '\n')

        {

            printf("\n");

        }

        else

        {

            printf("%c", c);

        }

}

}

*void* removepunct(FILE \*fp){

*char* str;

while((str = fgetc(fp)) != EOF){

// this line is checking if str is not a punctuation or whitespace character

if(!ispunct(str) && str!=' ' && str!='\n' && str!='\t' && str!='\v' && str!='\f' && str!='\r'){

str = lower(str);

printf("%c",str);

}

else{

// if whitespace or punctuation in encountered it prints newline

printf("\n");

}

}

}

*char* lower(*char* c){ // function to return the lowercase of passed argument

if(islower(c)){

return c;

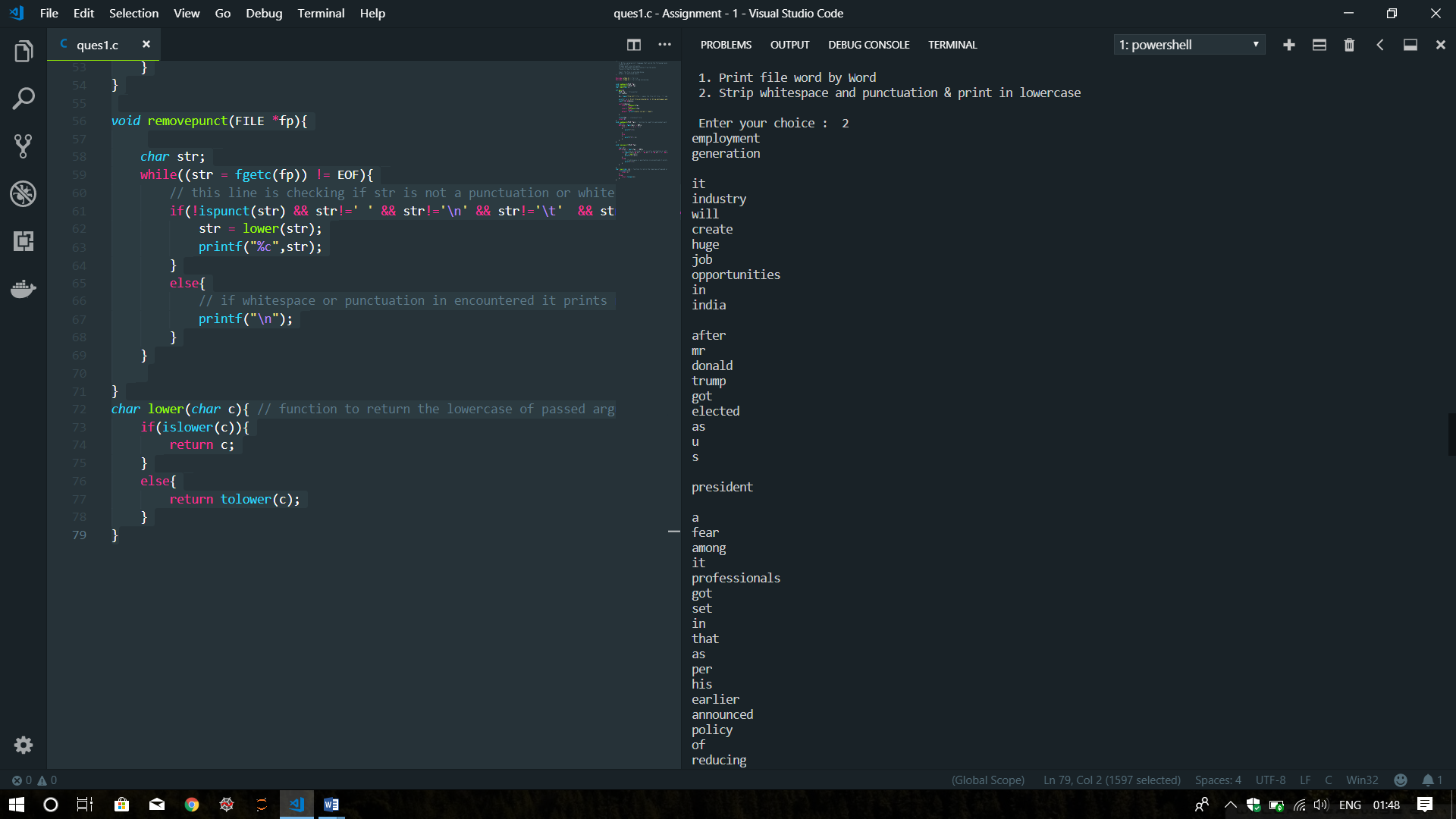
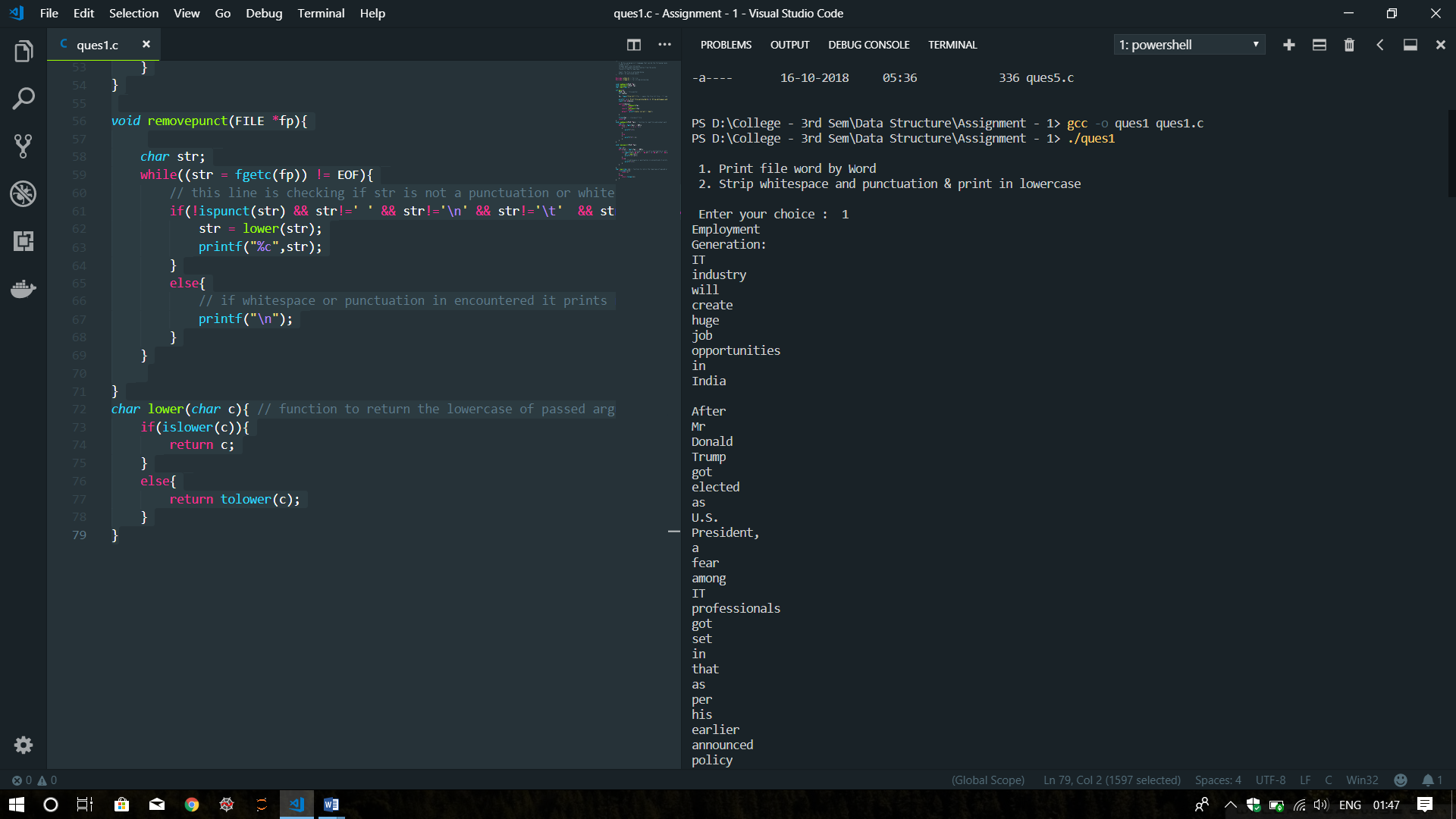
}

else{

return tolower(c);

}

}

**OUTPUT:**

2. There is an array given with the following elements: 20, 15, 25, 2, 30, 95, 41, 68, 4, 39. The array is an unsorted array. Write a program in C which searches the element 41 and 65 from the array with the total time complexity of O(n).

**Code:**

#include<stdio.h>

*int* LinearSearch(*int* array[], *int* item);

*int* main(*int* argc, *char* const \*argv[])

{

*int* array[10] = {20,15,25,2,30,95,41,68,4,39};

*int* loc = LinearSearch(array, 51);

if(loc!= -1)

printf("Found at : %d\n", loc);

else

printf("Element not found.");

return 0;

}

*int* LinearSearch(*int* array[], *int* item){

*int* i;

for(i=0 ; i<10; i++)

{

if(array[i] == item){

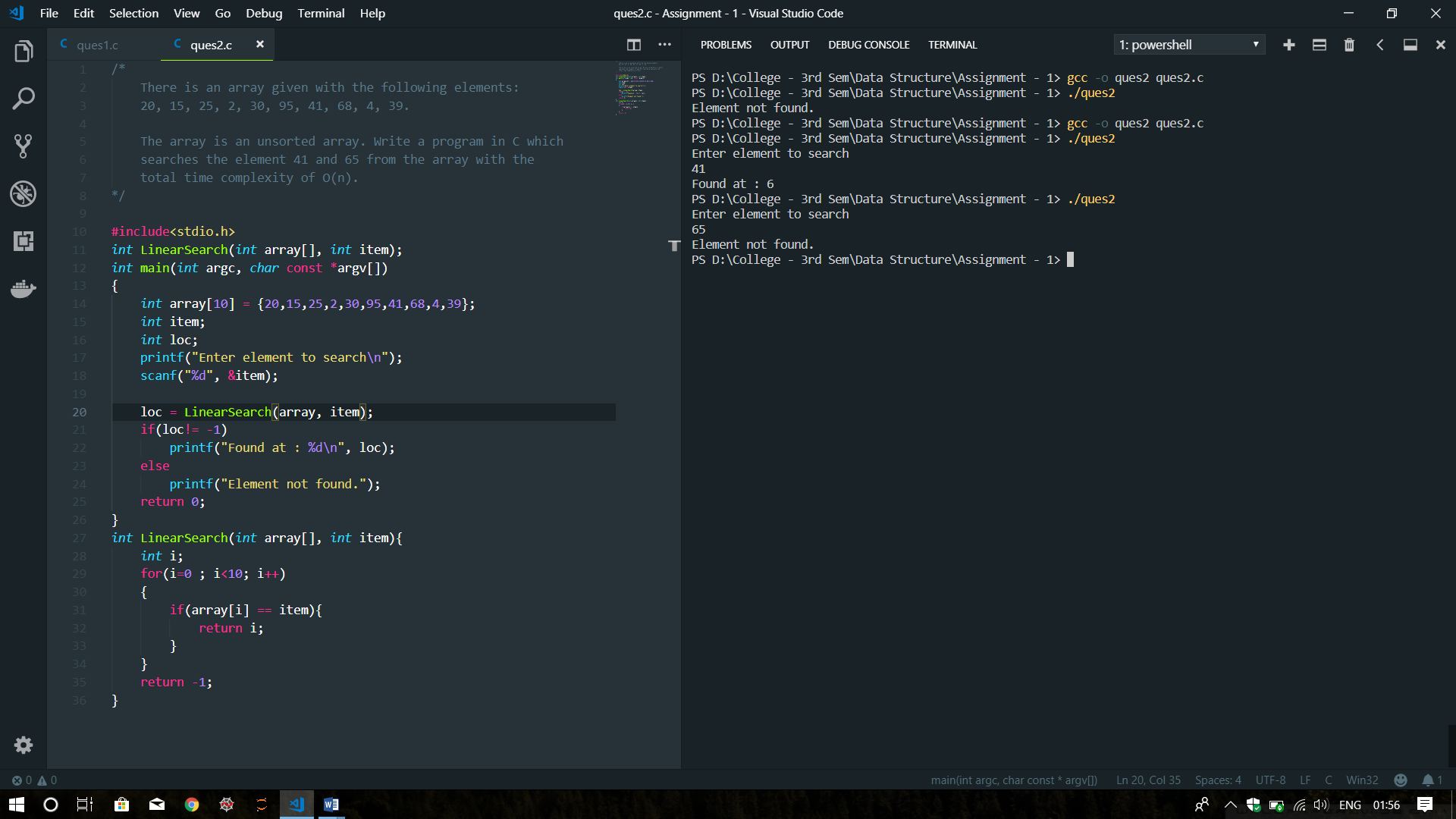
return i;

}

}

return -1;

}

**OUTPUT:**

3. There is an array of 12 elements. The elements are stored from A[2] to A[14] positions or locations in the memory. We need to delete an element from the array at the location A[7]. Write a program which delete the element at the location given and the deletion process must reduce the size of the array so that this chunk of memory can be used by other elements.

**Code:**

#include<stdio.h>

#include<stdlib.h>

*void* main()

{

//Initialize the array

*int* \*array;

*int* i,j;

*int* size;

*int* delete;

printf("Enter the size of array\n");

scanf("%d", &size);

array = (*int* \*)calloc(size,sizeof(*int*));

printf("Enter elements of array\n");

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

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

printf("Elements are: ");

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

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

printf("\n");

delete = array[3];

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

if(array[i] == delete){

for(j=i; j<size; j++){

array[j] = array[j+1];

}

}

}

//array = realloc(array, (size-1)\*sizeof(int));

printf("\nAfter deletion the array is:");

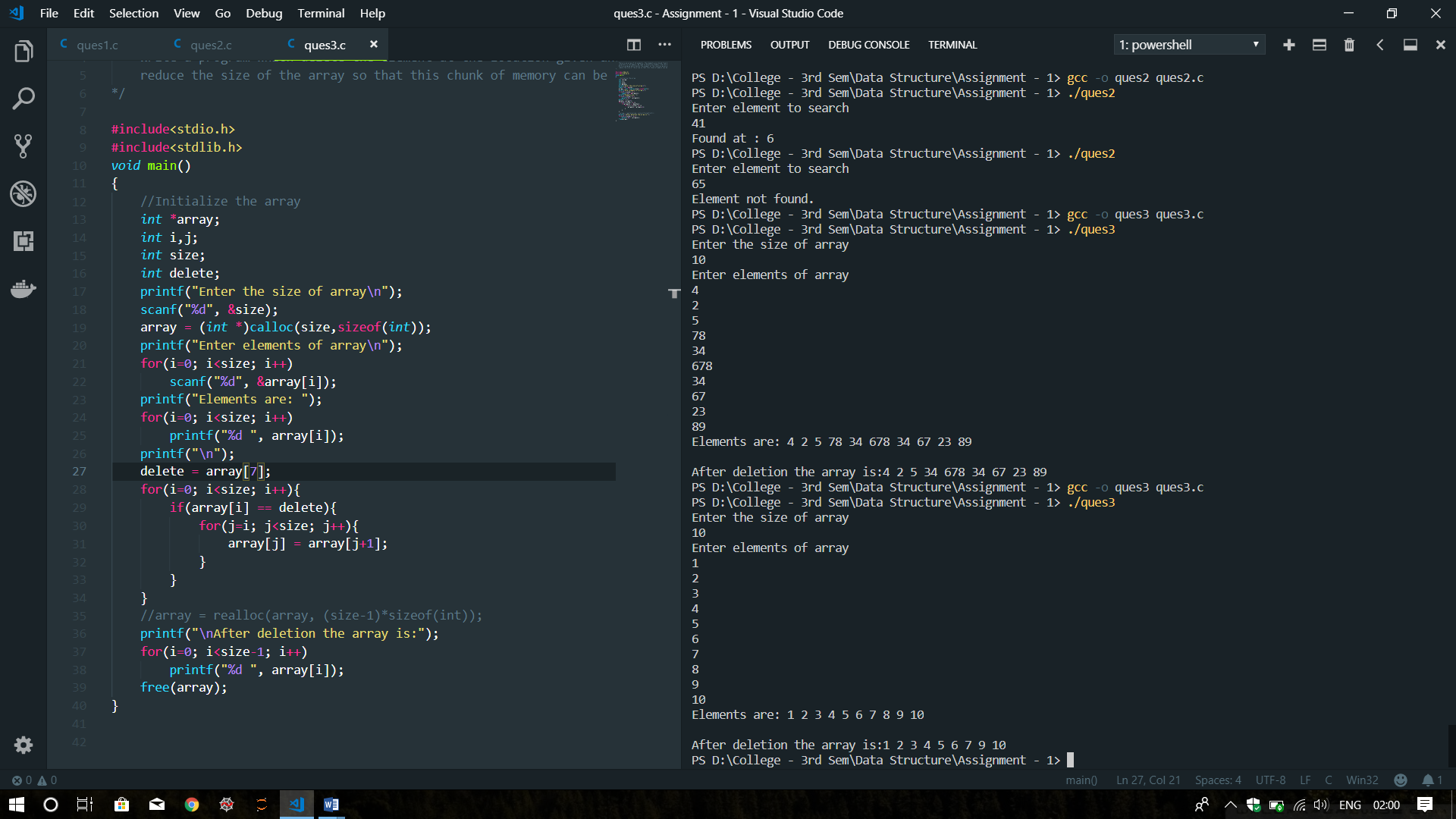
for(i=0; i<size-1; i++)

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

free(array);

}

**OUTPUT:**



4. Consider a link list with 6 elements. After inserting all the elements in the list, the elements are now taken out of the list one by one. The elements when all deleted are arranged in a way that the one which is inserted at first iteration comes at first spot and so on. Apply this first in first out strategy and display the result. The operations must be done in the following order:

* Insert()- insert the 6 elements in the list
* Delete()- delete elements one by one
* Find out the method in data structure which can use this delete() operation and apply a first in first out strategy
* Output will be an empty list.

**Code:**

#include<stdio.h>

#include<stdlib.h>

/\*Structure of node\*/

*struct* Node

{

*int* data;

*struct* Node \*next;

};

*void* printList(*struct* Node \*n)

{

while (n != NULL)

{

printf(" %d ", n->data);

n = n->next;

}

}

*void* InsertAtEnd(*struct* Node \*\*head, *int* item)

{

*struct* Node \* newnode = (*struct* Node \*)malloc(sizeof(*struct* Node));

newnode->data = item;

*struct* Node \*ptr = \*head;

if(\*head==NULL){

\*head = newnode;

newnode->next = NULL;

}

else{

while(ptr->next!=NULL){

ptr = ptr->next;

}

ptr->next = newnode;

newnode->next = NULL;

}

}

*void* DeleteAtBeg(*struct* Node\*\*head)

{

  if((\*head)==NULL)

  {

    printf("Underflow");

  }

  else{

*struct* Node \*ptr =\*head;

    \*head=ptr->next;

    free(ptr);

  }

}

// Main function

*int* main()

{

*struct* Node \*head;

//struct Node \* newnode;

head = (*struct* Node\*)malloc(sizeof(*struct* Node));

head = NULL;

/\* Inserting 6 elements \*/

InsertAtEnd(&head, 7);

InsertAtEnd(&head, 8);

InsertAtEnd(&head, 9);

InsertAtEnd(&head, 10);

InsertAtEnd(&head, 11);

InsertAtEnd(&head, 12);

/\* Deleting elements using FIFO operation \*/

DeleteAtBeg(&head);

DeleteAtBeg(&head);

DeleteAtBeg(&head);

DeleteAtBeg(&head);

DeleteAtBeg(&head);

DeleteAtBeg(&head);

/\* Output is empty list \*/

printList(head);

return 0;

}

**OUTPUT:**

