www.MyPlacementPrep.com

Pro Material Series

Question:25

You are required to fix all logical errors in the given code. You can click on Compile &amp; Run anytime to check the compilation/execution status of the program. You can use System.out.println to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the main() function as it is not required. Code Approach: For this question, you will need to correct the given implementation. We do not expect you to modify the approach or incorporate any additional library methods.

The method deleteDuplicate(intarr[]) of classDistinctArray takes an array as an input it is supposed to remove duplicates integers from the input array arr such that for each distinct integer the first occurrence is retained and all the duplicates elements following it are removed for Example given input array

(2,3,2,2,5,6,6,7)

the expected output is (2,3,5,6,7)

The function complies successfully but fails to return the desired results due to logical errors

Your task is debug the program to pass all the test cases

Corrected Code:

```
int* deleteDuplicate (int *arr, int *l)
{
        int len=*l;
        int i,j,k=0;
        for(i=0;i<len;i++)
        {
                for(j=i+1;j<len;j++)
                {
                        if(arr[j]==arr[i])
                        {
                                for(k=j;k<len-1;k++)
                                        arr[k]=arr[k+1];
                                len=len-1;
                                j--;
                        }
                }
        }
        *l=len;
        return arr;
}
```

FOR WORKING OUT

```
#include<stdio.h>
int * deleteDuplicate (int *, int *);
int main()
{
    int n,*arr,index;
    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*n);
    for(index=0;index<n;index++)
    {
            scanf("%d",&arr[index]);
    }
    arr=deleteDuplicate(arr,&n);
    for(index=0;index<n;index++)
    {
            printf("%d ",arr[index]);
    }
    return 0;

}
int* deleteDuplicate (int *arr, int *l)
{
            int len=*l;
            int i,j,k=0;
            for(i=0;i<len;i++)
            {
                        for(j=i+1;j<len;j++)
                        {
                                    if(arr[j]==arr[i])
                                    {
                                                for(k=j;k<len-1;k++)
                                                            arr[k]=arr[k+1];
                                                len=len-1;
                                                j--;
                                    }
                        }
            }
            *l=len;
            return arr;
}
```

QUESTION:26

The function sameelementcount(int *arr,intlen)accepts an integer array arr of length len as a input and returns the number of elements in an arr which are even numbers and equal to the element to its right

//WRITE DOWN YOUR CODE HERE

```
#include<stdio.h>
```

```
int sameelementcount(int *,int);
int main()
{
   int n,*arr,index;
   scanf("%d",&n);
   arr=(int *)malloc(sizeof(int)*n);
   for(index=0;index<n;index++)
   {
      scanf("%d",&arr[index]);
   }
   printf("%d",sameelementcount(arr,n));
   return 0;

}
int sameelementcount(int *arr, int len)
{
         int  i,count=0;
         for(i=0;i<len-1;i++)
         {
                  if((arr[i]%2==0)&&(arr[i]==arr[i+1]))
                           count++;
         }
         return  count;
}
```

QUESTION:27
Given a string str, write a program to eliminate all the vowels from the given string. The list of vowels in the English alphabet is : {a,e,i,o,u,A,E,l,0.U}. The Input to the function eliminateVowelString shall consist of a string str (containing only English letters) and returns a pointer to a string which does not contain vowels.
EXAMPLE:
Input =”abcdefghijklmnopqrstuvwxyz”
0utput="bcdfghjklmnpqrstvwxyz”
USEFUL COMMANDS:
strlen() is used to calculate the length of the string. The statement -
int len = strlen(str); Returns the length of the string str
TESTCASE 1:
Input: "bacdefghijklmnopgrstu"
Expected Return Value: "bcdfghjklmnpgrst"
TESTCASE 2:
Input: "bacdcfgh"
Expected Return Value: "bcdlgh"

```
#include<stdio.h>
char * removeVowel(char *);
```

```
int main()
{
    char *str="bacdefghijklmnopgrstu";
    printf("%s",removeVowel(str));
    return 0;

}
char * removeVowel(char *str)
{
            int trav,hold=0;
            for(trav=0;str[trav]!='\0';trav++)
            {
                        if(str[trav]=='a'||   str[trav]=='e'||   str[trav]=='i'||   str[trav]=='o'||   str[trav]=='u'||
str[trav]=='A'|| str[trav]=='E'|| str[trav]=='I'|| str[trav]=='O'|| str[trav]=='U')
                        {

                        }
                        else
                        {
                            str[hold]=str[trav];
                            hold++;
                        }
            }
            str[hold]='\0';
            return str;
}
```

QUESTION:28
Half sort Array:

```
#include<stdio.h>
#include<limits.h>
int main()
{
            int arr[]={10,12,25,6,13,8,19};
            int index,size,max,maxpos,min,minpos,temp,scope;
            size=sizeof(arr)/sizeof(arr[0]);
            for(index= 0 ; index < size; printf("%2d ",arr[index++]));
                        if(index%2==0)
                        {
                                    min = INT_MAX;
                                    for(index =1; index<size; index++)
                                    {
                                                if(arr[index] < min)
                                             {
```

```
                                        min = arr[index];
                                        minpos=index;
                                }
                        }
                        temp = arr[index];
                        arr[index] = arr[minpos];
                        arr[minpos]=temp;
                }
                else
                {
                        max = INT_MIN;
                        for(index = 0 ; index<size; index++)
                        {
                                if(arr[index] > max)
                                {
                                        max = arr[index];
                                        maxpos=index;
                                }
                        }
                        temp = arr[index];
                        arr[index] = arr[maxpos];
                        arr[maxpos]=temp;
                }
        for(printf("\n"),index= 0 ; index < size;printf("%2d ",arr[index++]));
        return 0;
}
```

Question : 30 WRITE YOUR CODE
Pyramid of alphabets

```
   a
  bcd
 efghi
jklmnop
```

```
#include<stdio.h>
void printPattern(int n)
{
        int i,j;
        char ch='a';
         for(i=1;i<=n;i++)
        {
           for(j=1;j<=n-i;j++)
              printf(" ");
           for(j=1;j<=2*i-1;j++)
              printf("%c",ch++);
           printf("\n");
```

```
            }

}
```

QUESTION:31

You have to encrypt a non-empty string phrase. The encryption adds a 'cyclic shift' to each letter where the value of this 'cyclic shift' is decided by the position of the letter from the end of its word. The shift value for each letter of a word is its index value (starting from 0) from the right-most character of the word.

EXAMPLE:

The shift values in 'yum feed' will be
yum: m->0, u->1, y->2
feed: d->0, e->1, e->2, f->3
which gives the encryption avmigfd
Here, adding the shift with value 0 to letter 'm' gives 'm' + 0 = m;
values 1 to 'u' gives 'u' + 1 = v and values 2 to 'y' gives 'y' + 2 = a and so on
Note that the shift wraps around on reaching the end of the alphabets, i.e., the shift values for 'y' as shown above is 'a'.

INPUT:

The input to the function/method consists of a string.

OUTPUT:

Return the encrypted string

NOTE:

Assume that the input string contains single space separating set of words

```
char * encryption(char* str);
int main()
{
   char str[13]="zebra tiger";
   printf("%s",encryption(str));
   return 0;
}
char  * encryption(char* str)
{
            //your CODE
            char nsubstr[10][10];
            int len,index,value,req_ind,i,j,count=0;
            for(i=0,j=0,index=0;str[index]!='\0';index++)
            {
                        if(str[index]==' ')
                        {
                                    nsubstr[i][j]='\0';
                                    j=0;
                        i++;
                        continue;
                }
                nsubstr[i][j++]=str[index];
        }
        count=i+1;;
        nsubstr[i][j]='\0';

        for(index=0,i=0;i<count;i++)
        {
                len=0;
                for(j=0;nsubstr[i][j]!='\0';j++)
                {
                        len++;

                }
                len--;
                for(j=0;nsubstr[i][j]!='\0';j++)
                {
                        if(nsubstr[i][j]+len<=122)
                                nsubstr[i][j]=nsubstr[i][j]+len--;
                        else
                                nsubstr[i][j]=nsubstr[i][j]+len-- -26;
                        str[index++]=nsubstr[i][j];
                }
                str[index++]=' ';

        }
        str[index]='\0';

        return str;
}
```

QUESTION:32

The LeastRecentlyUsed(LRU) cache algorithm exists the element from the cache(when it's full) that was leastrecentlyused. After an element is requested from the cache, it should be added to the cache(if not already there) and considered the most recently used element in the cache. Initially, the cache is empty. The input to the function LruCountMiss shall consist of an integer max_cache_size, an array pages and its length len. The function should return an integer for the number of cache misses using the LRU cache algorithm. Assume that the array pages always has pages numbered from 1 to 50.

TEST CASE1:

Input: 3 16 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0

Expected Return Value: 11
TESTCASE 2:
Input:2 9 2 3 1 3 2 1 4 3 2
Expected Return Value: 8

```
#include <stdio.h>
 int main()
{
   int max_cache_size,pages_len,i,j,cache[100],pages[100],k,pageincache=0,misscount=0;

   scanf("%d %d",&max_cache_size,&pages_len);
   for(i=0;i< max_cache_size;i++)
         cache[i]=-1;

   for(i=0;i<pages_len;i++)
   {
      pageincache=0;
      scanf("%d",&pages[i]);

      for(j=0;j<max_cache_size;j++)
      {
         if(pages[i]==cache[j])
         {
            pageincache=1;
            for(k=j;k<max_cache_size;k++)
            {
             cache[k]=cache[k+1];
            }
            cache[max_cache_size-1]=pages[i];
         }
      }
     if(pageincache==0)
     {
        misscount++;
        for(k=0;k<max_cache_size;k++)
        {
           cache[k]=cache[k+1];
        }
        cache[max_cache_size-1]=pages[i];
     }
   }
   printf("%d",misscount);
}
```