Lab Assingment-3

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 10/08/21

QUES 1: Given an unsorted dynamic array arr and two numbers x and y, find the minimum distance between x and y in arr. The array might also contain duplicates. You may assume that both x and y are different and present in arr.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

#define INFINITE 1000000

*int* between(*int*[], *int*, *int*, *int*);

*int* main()

{

*unsigned* *int* size;

    printf("Enter size of array: ");

    scanf("%d", &size);

*int* \*arr = (*int* \*)malloc(size \* sizeof(*int*));

    printf("Enter elements in array: ");

    for (*int* i = 0; i < size; i++)

        scanf("%d", &arr[i]);

*int* x, y;

    printf("Enter two elements: ");

    scanf("%d%d", &x, &y);

    printf("The minimum distance between the two elements: ");

    printf("%d\n", between(arr, size, x, y));

    free(arr);

    return 0;

}

*int* between(*int* *arr*[], *int* *size*, *int* *x*, *int* *y*)

{

*int* distance = INFINITE;

*int* i = 0, j = 1;

    while (i < *size*)

    {

        if (*arr*[i] == *x*)

        {

            while (*arr*[j] != *y* && j < *size*)

            {

                if (*arr*[j] == *x*)

                    i = j;

                j++;

            }

            if (*arr*[j] == *y* && j < *size*)

                distance = (j - i < distance) ? j - i : distance;

            i = j;

            continue;

        }

        else if (*arr*[i] == *y*)

        {

            while (*arr*[j] != *x* && j < *size*)

            {

                if (*arr*[j] == *y*)

                    i = j;

                j++;

            }

            if (*arr*[j] == *x* && j < *size*)

                distance = (j - i < distance) ? j - i : distance;

            i = j;

            continue;

        }

        i++;

    }

    return distance;

}

OUTPUT:

Enter size of array: 7

Enter elements in array: 1 3 5 4 2 7 6

Enter two elements: 3 7

The minimum distance between the two elements: 4

QUES 2: WAP to arrange the elements of a dynamic array such that all even numbers are followed by all odd numbers.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

*void* odd\_to\_even(*int* *arr*[], *int* *size*)

{

    for (*int* i = 1, j = 0; i < *size*; i++)

    {

        if (*arr*[i] % 2 == 0 && *arr*[j] % 2 == 1)

        {

*int* temp = *arr*[i];

*arr*[i] = *arr*[j];

*arr*[j] = temp;

            j++;

        }

        else if (*arr*[j] % 2 == 0)

            j++;

    }

}

*int* main()

{

*unsigned* *int* size;

    printf("Enter size of array: ");

    scanf("%d", &size);

*int* \*arr = (*int* \*)malloc(size \* sizeof(*int*));

    printf("Enter elements in array: ");

    for (*int* i = 0; i < size; i++)

        scanf("%d", &arr[i]);

    odd\_to\_even(arr, size);

    printf("Your arranged array: ");

    for (*int* i = 0; i < size; i++)

        printf("%d ", arr[i]);

    printf("\n");

    free(arr);

    return 0;

}

OUTPUT:

Enter size of array: 6

Enter elements in array: 1 8 9 4 5 2

Your arranged array: 8 4 2 1 5 9

QUES 3: Write a program to replace every element in the dynamic array with the next greatest element present in the same array.

SOLUTION

#include <stdio.h>

*int* main()

{

*int* a[] = {2, 5, 3, 9, 7};

*int* len = sizeof(a) / sizeof(a[0]);

    for (*int* i = 0; i < len; i++)

    {

        printf("%d ", a[i]);

    }

    printf("\n");

    for (*int* i = 0; i < len; i++)

    {

*int* max = a[i];

        for (*int* j = i + 1; j < len; j++)

        {

            if (max < a[j])

            {

                max = a[j];

                break;

            }

        }

        if (max == a[i])

            max = -1;

        a[i] = max;

    }

    for (*int* i = 0; i < len; i++)

    {

        printf("%d ", a[i]);

    }

}

OUTPUT:

2 5 3 9 7

5 9 9 -1 -1

QUES 4: WAP to sort rows of a dynamic matrix having m rows and n columns in ascending and columns in descending order.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

*void* sort\_row\_asc(*int* \*\*, *int*, *int*);

*void* sort\_column\_desc(*int* \*\*, *int*, *int*);

*int* \*\*create\_2d\_array(*int*, *int*);

*int* main()

{

*int* row, col;

    printf("Enter dimensions of matrix: ");

    scanf("%d%d", &row, &col);

*int* \*\*arr = create\_2d\_array(row, col);

    printf("Enter value in matrix:\n");

    for (*int* i = 0; i < row; i++)

        for (*int* j = 0; j < col; j++)

            scanf("%d", &arr[i][j]);

    sort\_row\_asc(arr, row, col);

    sort\_column\_desc(arr, row, col);

    printf("\nOutput:\n");

    for (*int* i = 0; i < row; i++)

    {

        for (*int* j = 0; j < col; j++)

            printf("%d ", arr[i][j]);

        printf("\n");

    }

    free(arr);

    return 0;

}

*void* sort\_row\_asc(*int* \*\**A*, *int* *row*, *int* *col*)

{

    for (*int* i = 0; i < *row*; i++)

    {

        for (*int* j = 0; j < *col* - 1; j++)

            for (*int* k = 0; k < *col* - j - 1; k++)

            {

                if (*A*[i][k + 1] < *A*[i][k])

                {

*int* temp = *A*[i][k + 1];

*A*[i][k + 1] = *A*[i][k];

*A*[i][k] = temp;

                }

            }

    }

}

*void* sort\_column\_desc(*int* \*\**A*, *int* *row*, *int* *col*)

{

    for (*int* i = 0; i < *col*; i++)

    {

        for (*int* j = 0; j < *row* - 1; j++)

            for (*int* k = 0; k < *row* - j - 1; k++)

            {

                if (*A*[k + 1][i] > *A*[k][i])

                {

*int* temp = *A*[k + 1][i];

*A*[k + 1][i] = *A*[k][i];

*A*[k][i] = temp;

                }

            }

    }

}

*int* \*\*create\_2d\_array(*int* *row*, *int* *col*)

{

*int* \*\*arr = (*int* \*\*)malloc(*row* \* sizeof(*int* \*));

    for (*int* i = 0; i < *row*; i++)

        arr[i] = (*int* \*)malloc(*col* \* sizeof(*int*));

    return arr;

}

OUTPUT:

Enter dimensions of matrix: 3 3

Enter value in matrix:

1 3 5

0 2 6

4 1 1

Output:

1 3 6

1 2 5

0 1 4

QUES 5: WAP to find out the k th smallest and k th largest element stored in a dynamic array of n integers, where 0<k<n.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

*void* bubble\_sort(*int*[], *int*);

*void* kth\_large\_and\_small(*int*[], *int*, *int*);

*int* main()

{

*int* size, k;

    printf("Enter size of array: ");

    scanf("%d", &size);

*int* \*arr = (*int* \*)malloc(size \* sizeof(*int*));

    for (*int* i = 0; i < size; i++)

        scanf("%d", &arr[i]);

    printf("Enter value of k: ");

    scanf("%d", &k);

    kth\_large\_and\_small(arr, size, k);

    free(arr);

    return 0;

}

*void* bubble\_sort(*int* *arr*[], *int* *size*)

{

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

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

        {

            if (*arr*[j + 1] < *arr*[j])

            {

*int* temp = *arr*[j + 1];

*arr*[j + 1] = *arr*[j];

*arr*[j] = temp;

            }

        }

}

*void* kth\_large\_and\_small(*int* *arr*[], *int* *size*, *int* *k*)

{

    bubble\_sort(*arr*, *size*);

    if (*k* < 1 || *k* > *size*)

    {

        printf("Error: K out of bounds!\n");

        return;

    }

    printf("%dth Largest: %d\n", *k*, *arr*[*size* - *k*]);

    printf("%dth Smallest: %d\n", *k*, *arr*[*k* - 1]);

}

OUTPUT:

Enter size of array: 5

7 2 5 4 1

Enter value of k: 4

4th Largest: 2

4th Smallest: 5

QUES 6: WAP to find the largest number and counts the occurrence of the largest number in a dynamic array of n integers using a single loop.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

*void* largest\_num\_and\_count(*int*[], *int*);

*int* main()

{

*int* size;

    printf("Enter size of array: ");

    scanf("%d", &size);

*int* \*arr = (*int* \*)malloc(size \* sizeof(*int*));

    printf("Enter values in array: ");

    for (*int* i = 0; i < size; i++)

        scanf("%d", &arr[i]);

    largest\_num\_and\_count(arr, size);

    free(arr);

    return 0;

}

*void* largest\_num\_and\_count(*int* *arr*[], *int* *size*)

{

*int* largest = *arr*[0];

*int* count = 1;

    for (*int* i = 1; i < *size*; i++)

    {

        if (largest < *arr*[i])

        {

            largest = *arr*[i];

            count = 1;

        }

        else if (largest == *arr*[i])

            count++;

    }

    printf("\nLargest Value: %d\n", largest);

    printf("Times present: %d\n", count);

}

OUTPUT:

Enter size of array: 5

Enter values in array: 1 7 7 3 6

Largest Value: 7

Times present: 2

QUES 7: You are given an array of 0 s and 1 s in random order. Segregate 0s on left side and 1s on right side of the array. Traverse array only once.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

*void* segg\_0s\_and\_1s(*int*[], *int*);

*int* main()

{

*int* size;

    printf("Enter size of array: ");

    scanf("%d", &size);

*int* \*arr = (*int* \*)malloc(size \* sizeof(*int*));

    for (*int* i = 0; i < size; i++)

        scanf("%d", &arr[i]);

    segg\_0s\_and\_1s(arr, size);

    for (*int* i = 0; i < size; i++)

        printf("%d ", arr[i]);

    printf("\n");

    free(arr);

    return 0;

}

*void* segg\_0s\_and\_1s(*int* *arr*[], *int* *size*)

{

*int* i = 0, j = *size* - 1;

    while (i < j)

    {

        if (*arr*[i] == 1 && *arr*[j] == 0)

        {

*arr*[i++] = 0;

*arr*[j--] = 1;

        }

        if (*arr*[j] == 1)

            j--;

        if (*arr*[i] == 0)

            i++;

    }

}

OUTPUT:

Enter size of array: 8

0 1 1 0 0 0 1 0

0 0 0 0 0 1 1 1

QUES 8: WAP to swap all the elements in the 1st column with all the corresponding elements in the last column, and 2nd column with the second last column and 3rd with 3rd last etc. of a 2-D dynamic array. Display the matrix.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

*void* row\_interchange(*int* \*\*, *int*, *int*);

*int* \*\*create\_2d\_array(*int*, *int*);

*void* input0\_output1(*int* \*\*, *int*, *int*, *int*);

*int* main()

{

*int* row, col;

    printf("Enter Rows and columns: ");

    scanf("%d%d", &row, &col);

*int* \*\*arr = create\_2d\_array(row, col);

    printf("Enter values in matrix:\n");

    input0\_output1(arr, row, col, 0);

    row\_interchange(arr, row, col);

    printf("\nOutput:\n");

    input0\_output1(arr, row, col, 1);

    free(arr);

    return 0;

}

*void* row\_interchange(*int* \*\**arr*, *int* *row*, *int* *col*)

{

    for (*int* j = 0, k = *col* - 1; j < *col* / 2; j++, k--)

        for (*int* i = 0; i < *row*; i++)

        {

*int* temp = *arr*[i][j];

*arr*[i][j] = *arr*[i][k];

*arr*[i][k] = temp;

        }

}

*int* \*\*create\_2d\_array(*int* *row*, *int* *col*)

{

*int* \*\*arr = (*int* \*\*)malloc(*row* \* sizeof(*int* \*));

    for (*int* i = 0; i < *row*; i++)

        arr[i] = (*int* \*)malloc(*col* \* sizeof(*int*));

    return arr;

}

*void* input0\_output1(*int* \*\**arr*, *int* *row*, *int* *col*, *int* *mode*)

{

    for (*int* i = 0; i < *row*; i++)

    {

        for (*int* j = 0; j < *col*; j++)

        {

            if (!*mode*)

            {

                scanf("%d", &*arr*[i][j]);

                continue;

            }

            printf("%d ", *arr*[i][j]);

        }

        if (*mode*)

            printf("\n");

    }

}

OUTPUT:

Enter Rows and columns: 3 3

Enter values in matrix:

1 2 3

4 5 6

7 8 9

Output:

3 2 1

6 5 4

9 8 7

QUES 9: WAP to store n employee’s data such as employee name, gender, designation, department, basic pay. Calculate the gross pay of each employees as follows:

Gross pay = basic pay + HRA + DA; HRA=25% of basic and DA=75% of basic.

SOLUTION:

#include <stdio.h>

#include <stdlib.h>

typedef *struct* employee

{

*char* name[50];

*char* gender;

*char* desig[50];

*char* dep[50];

*float* basic\_pay;

*float* gross\_pay;

} employee;

*void* output(employee);

*void* input(employee \*);

*int* main()

{

*int* size;

    printf("Enter the number of employees: ");

    scanf("%d", &size);

    employee \*E1 = (employee \*)malloc(size \* sizeof(employee));

    for (*int* i = 0; i < size; i++)

    {

        printf("\nEmployee number (%d):\n", i + 1);

        input(&E1[i]);

    }

    printf("\n");

    for (*int* i = 0; i < size; i++)

    {

        printf("Sl.No.:%d ", i + 1);

        output(E1[i]);

    }

    free(E1);

    return 0;

}

*void* output(employee *e1*)

{

    printf("%s\t", *e1*.name);

    printf("%c\t", *e1*.gender);

    printf("%s\t", *e1*.desig);

    printf("%s\t", *e1*.dep);

    printf("%0.2f\t", *e1*.basic\_pay);

    printf("%0.2f\n", *e1*.gross\_pay);

}

*void* input(employee \**e1*)

{

    printf("Enter Name: ");

    scanf(" %[^\n]s", (\**e1*).name);

    printf("Enter Gender: ");

    scanf(" %c", &(\**e1*).gender);

    printf("Enter designation: ");

    scanf(" %[^\n]s", (\**e1*).desig);

    printf("Enter department: ");

    scanf(" %[^\n]s", (\**e1*).dep);

    printf("Enter basic pay: ");

    scanf(" %f", &(\**e1*).basic\_pay);

*float* DA = 75 \* *e1*->basic\_pay;

*float* HRA = 25 \* *e1*->basic\_pay;

*e1*->gross\_pay = *e1*->basic\_pay + DA + HRA;

}

OUTPUT:

Enter the number of employees: 3

Employee number (1):

Enter Name: Sahil Singh

Enter Gender: M

Enter designation: Student

Enter department: CSE

Enter basic pay: 20000

Employee number (2):

Enter Name: Naam nahi pata

Enter Gender: F

Enter designation: Student

Enter department: CSE

Enter basic pay: 30000

Employee number (3):

Enter Name: Al Qaeda

Enter Gender: M

Enter designation: Terrorism

Enter department: ISIS

Enter basic pay: 65000000

Sl.No.:1 Sahil Singh    M       Student CSE     20000.00 2020000.00

Sl.No.:2 Naam nahi pata F       Student CSE     30000.00 3030000.00

Sl.No.:3 Al Qaeda       M       Terrorism       ISIS     65000000.00     6564999680.00

QUES 10: WAP to add two distances (in km-meter) by passing structure to a function.

SOLUTION:

#include <stdio.h>

typedef *struct* distance

{

*int* km;

*int* metre;

} distance;

distance add\_dist(distance, distance);

*int* main()

{

    distance dist1, dist2;

    printf("Enter distance1: ");

    scanf("%d%d", &dist1.km, &dist1.metre);

    printf("Enter distance 2: ");

    scanf("%d%d", &dist2.km, &dist2.metre);

    dist1 = add\_dist(dist1, dist2);

    printf("The resultant distance is: ");

    printf("%dKm %dM\n", dist1.km, dist1.metre);

    return 0;

}

distance add\_dist(distance *dist1*, distance *dist2*)

{

*dist1*.km = *dist1*.km + *dist2*.km;

*dist1*.km += (*dist1*.metre + *dist2*.metre) / 1000;

*dist1*.metre = (*dist1*.metre + *dist2*.metre) % 1000;

    return *dist1*;

}

OUTPUT:

Enter distance 1: 2 200

Enter distance 2: 3 200

The resultant distance is: 5Km 400M