/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q1.Write a C Program to store N elements to an array and then send all positive elements of the

array to the end without altering the original sequence.

\*/

#include <stdio.h>

#include <stdlib.h>

int main()

{

    int n;

    printf("Enter the size :");

    scanf("%d", &n);

    int ar[n];

    printf("Enter the array :\n");

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &ar[i]);

    }

    printf("Array is :\n");

    for (int i = 0; i < n; i++)

    {

        printf("%d  ", ar[i]);

    }

    printf("\n-----------OUTPUT-----------\n");

    printf("\nThe required array is :\n");

    int neg[n];

    int p = 0;

    for (int i = 0; i < n; i++)

    {

        if (ar[i] < 0)

        {

            neg[p++] = ar[i];

        }

    }

    for (int i = 0; i < n; i++)

    {

        if (ar[i] >= 0)

            neg[p++] = ar[i];

    }

    for (int i = 0; i < p; i++)

    {

        printf("%d  ", neg[i]);

    }

    return 0;

}

-----------OUTPUT-----------

enter the size of array 7

Enter the element

1

5

8

-1

4

-2

9

Modified array

-1 -2 8 1 4 5 9

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q2.Write a C program to create two linked lists positive and negative from a Original linked list,

so that positive linked list contains all positive elements and negative linked list contains

negative elements. Positive and negative linked lists should use the node of existing original

linked list.

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

    int data;

    struct node \*next;

} nodeType;

nodeType \*creatLL(nodeType \*);

void Positive\_and\_Negative(nodeType \*);

void display(nodeType \*);

int main()

{

    int c;

    nodeType \*head = NULL;

    while (1)

    {

        printf("\nenter the choice ");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            head = creatLL(head);

            break;

        case 2:

            Positive\_and\_Negative(head);

            exit(0);

            break;

        }

    }

}

nodeType \*creatLL(nodeType \*head)

{

    int c, val;

    nodeType \*newNode, \*temp;

    while (c)

    {

        newNode = (nodeType \*)malloc(sizeof(nodeType));

        printf("enter the data\t");

        scanf("%d", &val);

        newNode->data = val;

        newNode->next = NULL;

        if (head == NULL)

        {

            head = temp = newNode;

        }

        else

        {

            temp->next = newNode;

            temp = newNode;

        }

        printf("continue 1 or stop 0\t");

        scanf("%d", &c);

    }

    return head;

}

void Positive\_and\_Negative(nodeType \*head)

{

    nodeType \*p = NULL, \*q = NULL, \*temp, \*temp1;

    while (head != NULL)

    {

        if (head->data > 0)

        {

            nodeType \*new1 = (nodeType \*)malloc(sizeof(nodeType));

            new1->data = head->data;

            new1->next = NULL;

            if (p == NULL)

            {

                p = temp = new1;

            }

            else

            {

                temp->next = new1;

                temp = new1;

            }

        }

        else

        {

            nodeType \*new2 = (nodeType \*)malloc(sizeof(nodeType));

            new2->data = head->data;

            new2->next = NULL;

            if (q == NULL)

            {

                q = temp1 = new2;

            }

            else

            {

                temp1->next = new2;

                temp1 = new2;

            }

        }

        head = head->next;

    }

    printf("POSITIVE DATA LINKED LIST\n");

    display(p);

    printf("\n");

    printf("NEGATIVE DATA LINKED LIST\n");

    display(q);

}

void display(nodeType \*head)

{

    printf("[");

    while (head != NULL)

    {

        printf("%d ", head->data);

        head = head->next;

    }

    printf("]\n");

}

-----------OUTPUT-----------

enter the choice 1

continue 1 or stop 0 1

enter the data -8

continue 1 or stop 0 1

enter the data 9

continue 1 or stop 0 1

enter the data 3

continue 1 or stop 0 0

enter the choice 2

POSITIVE DATA LINKED LIST

[1 2 5 9 3 ]

NEGATIVE DATA LINKED LIST

[-4 -7 -8 ]

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q3.Write a C program to create a linked list P, and then write a ‘C’ function named split to

 create two linked lists Q & R from P So that Q contains all elements in odd positions of

 P and R contain the remaining elements. Finally print both linked lists i.e. Q and R.

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

    int data;

    struct node \*next;

} nodeType;

nodeType \*creatLL(nodeType \*);

void oddPositine\_and\_remaining(nodeType \*);

void display(nodeType \*);

int main()

{

    int c;

    nodeType \*head = NULL;

    while (1)

    {

        printf("\nenter the choice ");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            head = creatLL(head);

            break;

        case 2:

            oddPositine\_and\_remaining(head);

            exit(0);

            break;

        }

    }

}

nodeType \*creatLL(nodeType \*head)

{

    int c, val;

    nodeType \*newNode, \*temp;

    while (c)

    {

        newNode = (nodeType \*)malloc(sizeof(nodeType));

        printf("enter the data\t");

        scanf("%d", &val);

        newNode->data = val;

        newNode->next = NULL;

        if (head == NULL)

        {

            head = temp = newNode;

        }

        else

        {

            temp->next = newNode;

            temp = newNode;

        }

        printf("continue 1 or 0 stop \t");

        scanf("%d", &c);

    }

    return head;

}

void display(nodeType \*head)

{

    printf("[");

    nodeType \*p = head;

    while (p != NULL)

    {

        printf("%d ", p->data);

        p = p->next;

    }

    printf("]\n");

}

void oddPositine\_and\_remaining(nodeType \*head)

{

    nodeType \*p = NULL, \*q = NULL, \*temp, \*temp1;

    int i = 0;

    while (head != NULL)

    {

        if (i % 2 != 0)

        {

            nodeType \*new2 = (nodeType \*)malloc(sizeof(nodeType));

            new2->data = head->data;

            new2->next = NULL;

            if (q == NULL)

            {

                q = temp1 = new2;

            }

            else

            {

                temp1->next = new2;

                temp1 = new2;

            }

        }

        else

        {

            nodeType \*new1 = (nodeType \*)malloc(sizeof(nodeType));

            new1->data = head->data;

            new1->next = NULL;

            if (p == NULL)

            {

                p = temp = new1;

            }

            else

            {

                temp->next = new1;

                temp = new1;

            }

        }

        i++;

        head = head->next;

    }

    printf("ODD POSITION LIST\n");

    display(q);

    printf("\n");

    printf("REMAINING POSITION LIST\n");

    display(p);

}

-----------OUTPUT-----------

enter the choice 1

enter the data 1

continue 1 or 0 stop 1

enter the data 2

continue 1 or 0 stop 1

enter the data 3

continue 1 or 0 stop 1

enter the data 4

continue 1 or 0 stop 1

enter the data 5

continue 1 or 0 stop 1

enter the data 6

continue 1 or 0 stop 1

enter the data 7

continue 1 or 0 stop 1

enter the data 8

continue 1 or 0 stop 0

enter the choice 2

ODD POSITION LIST

[1 3 5 7 ]

REMAINING POSITION LIST

[2 4 6 8 ]

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q4.Write a C program to add of two polynomials of degree n, using linked list

 For example p1=anx n+an-1x n-1+ an-2x n-2+ …….. a0x 0

 P2=bnx n+bn-1x n-1+ bn-2x n-2+ …….. b0x 0

 p1 = first polynomial

 p2 = second polynomial

 Find out p3= p1+p2

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

    int coefficient;

    int exponent;

    struct Node \*next;

} Node;

void insert(Node \*\*head, int coefficient, int exponent)

{

    Node \*new\_node = (Node \*)malloc(sizeof(Node));

    Node \*temp;

    temp = (\*head);

    new\_node->coefficient = coefficient;

    new\_node->exponent = exponent;

    if (temp == NULL)

    {

        \*head = temp = new\_node;

    }

    else

    {

        while (temp->next != NULL)

        {

            temp = temp->next;

        }

        temp->next = (new\_node);

        temp = new\_node;

    }

    temp->next = NULL;

}

Node \*add(Node \*p1, Node \*p2)

{

    Node \*result = NULL;

    Node \*temp1 = p1, \*temp2 = p2;

    while (temp1 != NULL && temp2 != NULL)

    {

        if (temp1->exponent > temp2->exponent)

        {

            insert(&result, temp1->coefficient, temp1->exponent);

            temp1 = temp1->next;

        }

        else if (temp1->exponent < temp2->exponent)

        {

            insert(&result, temp2->coefficient, temp2->exponent);

            temp2 = temp2->next;

        }

        else

        {

            insert(&result, temp1->coefficient + temp2->coefficient, temp1->exponent);

            temp1 = temp1->next;

            temp2 = temp2->next;

        }

    }

    // Add the remaining terms from p1, if any

    while (temp1 != NULL)

    {

        insert(&result, temp1->coefficient, temp1->exponent);

        temp1 = temp1->next;

    }

    // Add the remaining terms from p2, if any

    while (temp2 != NULL)

    {

        insert(&result, temp2->coefficient, temp2->exponent);

        temp2 = temp2->next;

    }

    return result;

}

// Function to print the polynomial in the form of ax^b + cx^d +...

void printPolynomial(Node \*head)

{

    printf("[");

    while (head != NULL)

    {

        if (head->exponent == 0)

            printf("%d", head->coefficient);

        else

            printf("%dx^%d+", head->coefficient, head->exponent);

        head = head->next;

    }

    printf("]\n");

}

int main()

{

    int c, coeff, expo;

    Node \*head = NULL, \*head1 = NULL, \*result = NULL;

    while (1)

    {

        printf("\n1.enter the coefficient and eponent of 1st polynomial list\n2.enter the coefficient and eponent of 2nd polynomial List\n3.adding list\n4.Print Added list\n5.exit\nenter the choice  ");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            printf("\nenter the coefficient and eponent\n");

            scanf("%d %d", &coeff, &expo);

            insert(&head, coeff, expo);

            break;

        case 2:

            printf("\nenter the coefficient and eponent\n");

            scanf("%d %d", &coeff, &expo);

            insert(&head1, coeff, expo);

            break;

        case 3:

            printf("\nOPERATION DONE OF ADDING\n");

            result = add(head, head1);

            break;

        case 4:

            printf("\nADDED FINAL LIST\n");

            printPolynomial(result);

            break;

        case 5:

            exit(0);

        default:

            break;

        }

    } }

-----------OUTPUT-----------

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 1

enter the coefficient and exponent of 1st LL

2

3

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 1

enter the coefficient and exponent of 1st LL

2

2

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 1

enter the coefficient and exponent of 1st LL

3

1

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 1

enter the coefficient and exponent of 1st LL

5

0

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 2

enter the coefficient and exponent of 2nd LL

2

1

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 2

enter the coefficient and exponent of 2nd LL

7

0

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 3

OPERATION DONE OF ADDING

1.enter the coefficient and exponent of 1st polynomial list

2.enter the coefficient and exponent of 2nd polynomial List

3.adding list

4.Print Added list

5.exit

enter the choice 4

ADDED FINAL LIST

[2x^3+2x^2+5x^1+12]

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q6.Write a C program to create a double linked list by inserting nodes in such a way that the

 resultant linked list remains in ascending order.(do not use any sorting technique).

\*/

#include <stdio.h>

#include <stdlib.h>

struct Node

{

  int data;

  struct Node \*prev;

  struct Node \*next;

};

struct Node \*createNode(int data)

{

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

  node->data = data;

  node->prev = NULL;

  node->next = NULL;

  return node;

}

void insertNode(struct Node \*\*head, int data)

{

  struct Node \*node = createNode(data);

  if (\*head == NULL)

  {

    \*head = node;

    return;

  }

  if (node->data <= (\*head)->data)

  {

    node->next = \*head;

    (\*head)->prev = node;

    \*head = node;

    return;

  }

  struct Node \*current = \*head;

  while (current->next != NULL && current->next->data < node->data)

  {

    current = current->next;

  }

  node->next = current->next;

  if (current->next != NULL)

  {

    current->next->prev = node;

  }

  current->next = node;

  node->prev = current;

}

void printList(struct Node \*head)

{

  printf("[");

  while (head != NULL)

  {

    printf(" %d ", head->data);

    head = head->next;

  }

  printf("]\n");

}

int main()

{

  struct Node \*head = NULL;

  int c, val;

  while (1)

  {

    printf("\nenter the choice ");

    scanf("%d", &c);

    switch (c)

    {

    case 1:

      printf("enter the val\t");

      scanf("%d", &val);

      insertNode(&head, val);

      break;

    case 2:

      printList(head);

      exit(0);

    default:

      break;

    }

  }

  return 0;}

-----------OUTPUT-----------

enter the choice 1

enter the val 1

enter the choice 1

enter the val 4

enter the choice 1

enter the val 6

enter the choice 1

enter the val 8

enter the choice 1

enter the val 7

enter the choice 1

enter the val 3

enter the choice 2

[ 1 3 4 6 7 8 ]

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q7. Write a C program to create a binary search tree and perform following operations:

 1) Search a particular key.

 2) Delete a node from the tree.

 3) Find total number of leaf nodes

 4) Find height of a binary search tree

 5) Count total numbers of nodes from right hand side of root node

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct tree

{

    struct tree \*left;

    int data;

    struct tree \*right;

} treeType;

treeType \*bst(treeType \*);

treeType \*creat(int);

void insert(treeType \*, treeType \*);

int totalLeafNodes(treeType \*);

void inorder(treeType \*);

int height(treeType \*);

treeType \*search(treeType \*, int);

treeType \*smallest(treeType \*root);

treeType \*deleteNode(treeType \*, int );

static int count = 0;

int main()

{

    int num, val, c, key;

    treeType \*root = NULL, \*result;

    while (1)

    {

        printf("\nenter the choice\n1.creat Tree\n2.Total Leaf Node\n3.Inorder print\n4.Height Of Tree\n5.Search Key\n6.delete Node\n7.total numbers of nodes from right hand side of root node\n8.exit\n");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            root = bst(root);

            break;

        case 2:

            printf("\nTotal number of Leaf nodes are %d\n", totalLeafNodes(root));

            break;

        case 3:

            inorder(root);

            break;

        case 4:

            printf("\nheight of tree is %d\n", height(root));

            break;

        case 5:

            printf("\nenter the key\t");

            scanf("%d", &key);

            result = search(root, key);

            if (result != NULL)

            {

                printf("the key is present\n");

            }

            else

            {

                printf("key not present\n");

            }

            break;

        case 6:

            printf("\nenter the value to be deleted\t");

            scanf("%d", &val);

            root = deleteNode(root, val);

            break;

        case 7:

            exit(0);

        default:

            break;

        }

    }

}

treeType \*bst(treeType \*root)

{

    treeType \*temp;

    int c, num;

    while (c)

    {

        printf("enter the data");

        scanf("%d", &num);

        temp = creat(num);

        if (root == NULL)

        {

            root = temp;

        }

        else

        {

            insert(root, temp);

        }

        printf("do u want to continue 1 or 0\t");

        scanf("%d", &c);

    }

    return root;

}

treeType \*creat(int num)

{

    treeType \*newNode = (treeType \*)malloc(sizeof(treeType));

    if (newNode == NULL)

    {

        printf("memory not allocated");

    }

    else

    {

        newNode->left = NULL;

        newNode->data = num;

        newNode->right = NULL;

    }

    return newNode;

}

void insert(treeType \*root, treeType \*temp)

{

    if (root->data == temp->data)

    {

        printf("no insertion \n");

    }

    else if (temp->data < root->data)

    {

        if (root->left == NULL)

            root->left = temp;

        else

            insert(root->left, temp);

    }

    else

    {

        if (root->right == NULL)

            root->right = temp;

        else

            insert(root->right, temp);

    }

}

int totalLeafNodes(treeType \*root)

{

    if (root == NULL)

        return 0;

    else if (root->left == NULL && root->right == NULL)

        return 1;

    else

        return totalLeafNodes(root->left) + totalLeafNodes(root->right);

}

void inorder(treeType \*root)

{

    int c;

    if (root != NULL)

    {

        inorder(root->left);

        printf("%d\t", root->data);

        inorder(root->right);

    }

}

int height(treeType \*root)

{

    int leftheight, rightheight;

    if (root == NULL)

        return 0;

    else

    {

        leftheight = height(root->left);

        rightheight = height(root->right);

        if (leftheight > rightheight)

        {

            return (leftheight + 1);

        }

        else

        {

            return (rightheight + 1);

        }

    }

}

treeType \*search(treeType \*root, int key)

{

    if (root == NULL)

    {

        return 0;

    }

    if (root->data == key)

        return root;

    else

    {

        if (key < root->data)

        {

            return search(root->left, key);

        }

        else

        {

            return search(root->right, key);

        }

    }

}

treeType \*deleteNode(treeType \*root, int val)

{

    // base case

    if (root == NULL)

        return root;

    // If the key to be deleted is smaller than the root's key,

    // then it lies in left subtree

    if (val < root->data)

    {

        root->left = deleteNode(root->left, val);

    } // If the key to be deleted is greater than the root's key,

    // then it lies in right subtree

    else if (val > root->data)

    {

        root->right = deleteNode(root->right, val);

    }

    // if key is same as root's key, then This is the node

    // to be deleted

    else

    {

        // node with only one child or no child

        if (root->left == NULL)

        {

            treeType \*temp = root->right;

            free(root);

            return temp;

        }

        else if (root->right == NULL)

        {

            treeType \*temp = root->left;

            free(root);

            return temp;

        }

        treeType \*temp = smallest(root->right);

        root->data = temp->data;

        root->right = deleteNode(root->right, temp->data);

    }

    return root;

}

treeType \*smallest(treeType \*root)

{

    treeType \*cur = root;

    while (cur->left != NULL)

    {

        cur = cur->left;

    }

    return cur;

}

-----------OUTPUT-----------

enter the choice 1

enter the data99

do u want to continue 1 or 0 1

enter the data50

do u want to continue 1 or 0 1

enter the data200

do u want to continue 1 or 0 1

enter the data150

do u want to continue 1 or 0 1

enter the data170

do u want to continue 1 or 0 1

enter the data25

do u want to continue 1 or 0 0

enter the choice 2

Total number of Leaf nodes are 2

enter the choice 3

25 50 99 150 170 200

enter the choice 4

height of tree is 4

enter the choice 5

enter the key 50

the key is present

enter the choice 6

enter the value to be deleted 200

enter the choice 3

25 50 99 150 170

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q8.Write a C program to sort an unsorted sequence of strings given by user in an array, using

 Merge sort technique.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 10

void Merge(char \*arr[], int low, int mid, int high)

{

    int nL = mid - low + 1;

    int nR = high - mid;

    char \*\*L = malloc(sizeof(char \*) \* nL);

    char \*\*R = malloc(sizeof(char \*) \* nR);

    int i;

    for (i = 0; i < nL; i++)

    {

        L[i] = malloc(sizeof(arr[low + i]));

        strcpy(L[i], arr[low + i]);

    }

    for (i = 0; i < nR; i++)

    {

        R[i] = malloc(sizeof(arr[mid + i + 1]));

        strcpy(R[i], arr[mid + i + 1]);

    }

    int j = 0, k;

    i = 0;

    k = low;

    while (i < nL && j < nR)

    {

        if (strcmp(L[i], R[j]) < 0)

            strcpy(arr[k++], L[i++]);

        else

            strcpy(arr[k++], R[j++]);

    }

    while (i < nL)

        strcpy(arr[k++], L[i++]);

    while (j < nR)

        strcpy(arr[k++], R[j++]);

}

void MergeSort(char \*arr[], int low, int high)

{

    if (low < high)

    {

        int mid = (low + high) / 2;

        MergeSort(arr, low, mid);

        MergeSort(arr, mid + 1, high);

        Merge(arr, low, mid, high);

    }

}

int main()

{

    int size;

    printf("\nEnter the size of array : ");

    scanf("%d", &size);

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

    printf("\nEnter the strings of the array: ");

    int i;

    for (i = 0; i < size; i++)

    {

        arr[i] = malloc(sizeof(char) \* MAX);

        printf("\nEnter String: ");

        scanf("%s", arr[i]);

    }

    MergeSort(arr, 0, size - 1);

    printf("\n-----------OUTPUT-----------\n");

    printf("\nThe Sorted Array of String is\n");

    for (i = 0; i < size; i++)

        printf("%s ", arr[i]);

    return 0;

}

-----------OUTPUT-----------

Enter the size of array : 5

Enter the strings of the array:

Enter String: harsh

Enter String: vipul

Enter String: aditya

Enter String: vikas

Enter String: seema

The Sorted Array of String is

aditya harsh seema vikas Vipul

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q9.Write a C program to implement linked representation of a graph in memory using array of

 pointers

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

typedef struct Graph

{

    int data;

    struct Graph \*next;

} GraphType;

void CreatGraph(GraphType \*arr[], int vertice)

{

    GraphType \*p, \*temp;

    int edges, val;

    for (int i = 0; i < vertice; i++)

    {

        temp = NULL;

        printf("\nEnter the no of edges : ");

        scanf("%d", &edges);

        for (int j = 0; j < edges; j++)

        {

            p = (GraphType \*)malloc(sizeof(GraphType));

            printf("Enter the neighbour of %d ", i);

            scanf("%d", &val);

            p->data = val;

            p->next = NULL;

            if (arr[i] == NULL)

                arr[i] = p;

            else

                temp->next = p;

            temp = p;

        }

    }

}

void printGraph(GraphType \*arr[], int vertice)

{

    GraphType \*temp;

    for (int i = 0; i < vertice; i++)

    {

        temp = arr[i];

        printf("\nThe neighbour of %d is", i);

        while (temp != NULL)

        {

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

            temp = temp->next;

        }

    }

}

int main()

{

    int val, edges, vertice;

    printf("Enter the no of vertices");

    scanf("%d", &vertice);

    GraphType \*arr[vertice];

    for (int i = 0; i < vertice; i++)

    {

        arr[i] = NULL;

    }

    CreatGraph(arr, vertice);

    printf("\n-----------OUTPUT-----------\n");

    printGraph(arr, vertice);

}

-----------OUTPUT-----------

Enter the no of vertices 4

Enter the no of edges of 0: 2

Enter the neighbour of 0 1

Enter the neighbour of 0 2

Enter the no of edges of 1: 3

Enter the neighbour of 1 2

Enter the neighbour of 1 3

Enter the neighbour of 1 4

Enter the no of edges of 2: 2

Enter the neighbour of 2 3

Enter the neighbour of 2 4

Enter the no of edges of 3: 2

Enter the neighbour of 3 0

Enter the neighbour of 3 4

The neighbour of 0 is-> 1-> 2

The neighbour of 1 is-> 2-> 3-> 4

The neighbour of 2 is-> 3-> 4

The neighbour of 3 is-> 0-> 4

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q10.Write a C program to implement DFS.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100

int adj[MAX][MAX];

int visited[MAX];

void create\_graph(int);

void DFS(int, int);

void display(int);

int main()

{

    int v, n;

    printf("Enter number of vertices : ");

    scanf("%d", &n);

    create\_graph(n);

    display(n);

    printf("Enter the starting vertex for Depth First Search : ");

    scanf("%d", &v);

    DFS(v, n);

    return 0;

}

void DFS(int v, int n)

{

    int i;

    visited[v] = 1;

    printf("%d ", v);

    for (i = 1; i <= n; i++)

    {

        if (adj[v][i] == 1 && visited[i] == 0)

            DFS(i, n);

    }

}

void create\_graph(int n)

{

    int i, max\_edges, origin, destin;

    max\_edges = n \* (n - 1);

    for (i = 1; i <= max\_edges; i++)

    {

        printf("Enter edge %d( -1 -1 to quit ) : ", i);

        scanf("%d %d", &origin, &destin);

        if ((origin == -1) && (destin == -1))

            break;

        if (origin > n || destin > n || origin < 0 || destin < 0)

        {

            printf("Invalid edge!\n");

            i--;

        }

        else

        {

            adj[origin][destin] = 1;

        }

    }

}

void display(int n)

{

    int i, j;

    for (i = 1; i <= n; i++)

    {

        for (j = 1; j <= n; j++)

            printf("%4d", adj[i][j]);

        printf("\n");

    }

}

-----------OUTPUT-----------

Enter number of vertices : 4

Enter edge 1( -1 -1 to quit ) : 1 2

Enter edge 2( -1 -1 to quit ) : 1 3

Enter edge 3( -1 -1 to quit ) : 2 3

Enter edge 4( -1 -1 to quit ) : 3 4

Enter edge 5( -1 -1 to quit ) : 1 4

Enter edge 6( -1 -1 to quit ) : -1 -1

0 1 1 1

0 0 1 0

0 0 0 1

0 0 0 0

Enter the starting vertex for Depth First Search : 2

2 3 4

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q5.Wriet a C program to implement time sharing environment (using circular linked list) for N

 processes, where CPU allocates time slots of 10ns for given N processes, then print which

 process will be completed in how much time

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct nodeType

{

    struct nodeType \*next;

    int info;

    int sno;

}nodeType;

nodeType\* insert(nodeType \*start,int n)

{

    static int sno=0;

    nodeType \*ptr=(nodeType\*)malloc(sizeof(nodeType));

    ptr->sno=++sno;

    ptr->info=n;

    if(start==NULL)

        ptr->next=ptr;

    else

    {

        ptr->next=start->next;

        start->next=ptr;

    }

    return ptr;

}

void traverse(nodeType\* start)

{

    if (start==NULL)

        printf("No data available\n");

    else

    {

        nodeType \*t=start;

        printf("Time Taken\n");

        while(t->next!=start)

        {

            printf("%d\t", t->info);

            t=t->next;

        }

        printf("%d\n", t->info);

    }

}

void delete(nodeType \*\*p)

{

    nodeType \*q=\*p, \*r=NULL;

    if(q->next==q)

    {

        free(q);

        \*p=NULL;

    }

    else

    {

        r=q->next;

        q->next=r->next;

        free(r);

        \*p=q;

    }

}

void taskprocess(nodeType \*\*p, int timee)

{

    int c=0, y;

    nodeType \*q=\*p;

    while(q)

    {

        y=(++c)\*timee;

        nodeType \*f=q->next;

        f->info-=timee;

        if((f->info)<=0)

        {

            printf("Process-%d is completed in %d turn within %d unit time\n", f->sno, c, y);

            delete(&q);

        }

        else q=q->next;

    }

}

int main()

{

    nodeType \*top=NULL;

    int choice, x, ch=1;

    printf("OPERATION YOU NEED TO PERFORM:\n1. INSERT\n2. PROCESS SCHEDULING\n3. DISPLAY\n4. EXIT\n");

    do

    {

        printf("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

                printf("Enter time: ");

                scanf("%d", &x);

                top=insert(top, x);

                break;

            case 2:

                taskprocess(&top, 10);

                break;

            case 3:

                traverse(top);

                break;

            case 4:

                ch=0;

                break;

            default:

                printf("Invalid choice hence exit\n");

                break;

        }

    } while(ch==1);

}

-----------OUTPUT-----------

OPERATION YOU NEED TO PERFORM:

1. INSERT

2. PROCESS SCHEDULING

3. DISPLAY

4. EXIT

Enter your choice: 1

Enter time: 20

Enter your choice: 1

Enter time: 30

Enter your choice: 1

Enter time: 50

Enter your choice: 1

Enter time: 40

Enter your choice: 3

Time Taken

40 20 30 50

Enter your choice: 2

Process-1 is completed in 5 turn within 50 unit time

Process-2 is completed in 9 turn within 90 unit time

Process-4 is completed in 13 turn within 130 unit time

Process-3 is completed in 14 turn within 140 unit time

Enter your choice: 4

/\*

NAME : Aayush Kumar

ROLL.NO : 2019284

SECTION : B

Q11.Write a C program to implement Kurskal’s algorithm.

\*/

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int parent[9];

int find(int);

int uni(int, int);

void main()

{

    int i, j, k, a, b, u, v, n, ne = 1;

    int min, mincost = 0, cost[9][9];

    printf("Implementation of Kruskal's algorithm\n");

    printf("\nEnter the no. of vertices:");

    scanf("%d", &n);

    printf("\nEnter the cost adjacency matrix:\n");

    for (i = 1; i <= n; i++)

    {

        for (j = 1; j <= n; j++)

        {

            scanf("%d", &cost[i][j]);

            if (cost[i][j] == 0)

                cost[i][j] = 999;

        }

    }

    printf("The edges of Minimum Cost Spanning Tree are\n");

    while (ne < n)

    {

        for (i = 1, min = 999; i <= n; i++)

        {

            for (j = 1; j <= n; j++)

            {

                if (cost[i][j] < min)

                {

                    min = cost[i][j];

                    a = u = i;

                    b = v = j;

                }

            }

        }

        u = find(u);

        v = find(v);

        if (uni(u, v))

        {

            printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);

            mincost += min;

        }

        cost[a][b] = cost[b][a] = 999;

    }

    printf("\nMinimum cost = %d\n", mincost);

}

int uni(int i, int j)

{

    if (i != j)

    {

        parent[j] = i;

        return 1;

    }

    return 0;

}

int find(int i)

{

    while (parent[i])

        i = parent[i];

    return i;

}

-----------OUTPUT-----------

Enter the no. of vertices:3

Enter the cost adjacency matrix:

1

2

3

6

7

3

1

8

9

The edges of Minimum Cost Spanning Tree are

1 edge (3,1) =1

2 edge (1,2) =2

Minimum cost = 3