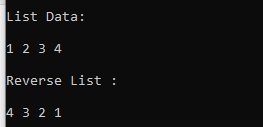
**Lab 5 DS**

**K226007**

**Syed Yousha Mehdi**

**BSR-3C**

Task1:



#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int d) : data(d), next(NULL) {}

};

// struct Node

// {

// int data;

// Node \*next;

// };

class Singli

{

public:

Node \*head, \*tail;

Singli()

{

head = NULL;

tail = NULL;

}

void insert\_at\_tail(int num)

{

Node \*temp = new Node(num);

if (head == NULL)

{

head = temp;

tail = temp;

}

else

{

tail->next = temp;

tail = tail->next;

tail->next = NULL;

}

}

void display()

{

Node \*temp = head;

cout << "\nList Data: \n\n";

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

}

void reverse(Node \*temp)

{

if (temp == NULL)

return;

reverse(temp->next);

cout << temp->data << " ";

}

};

int main()

{

Singli obj;

obj.insert\_at\_tail(1);

obj.insert\_at\_tail(2);

obj.insert\_at\_tail(3);

obj.insert\_at\_tail(4);

obj.display();

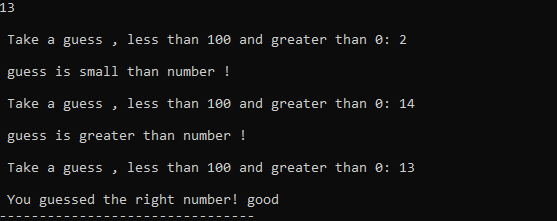
cout<<"\n\nReverse List : \n\n";

obj.reverse(obj.head);

return 0;

}

Task2:



#include <iostream>

#include <cstdlib>

#include <time.h>

using namespace std;

void guess(int num, bool flag)

{

int static n = num;

if(n == num && flag == 1)

{

cout<<"\n You guessed the right number! good";

exit(0);

}

else if(num < n)

{

cout<<"\n guess is small than number !";

}

else if(num > n)

{

cout<<"\n guess is greater than number !";

}

jump:

int guess\_num;

cout<<"\n\n Take a guess , less than 100 and greater than 0: ";

cin>> guess\_num;

if( guess\_num < 0)

{

cout<<"\error! number should be greater than 0!";

goto jump;

}

if( guess\_num > 100)

{

cout<<"\error! number should be smaller than 100!";

goto jump;

}

guess(guess\_num, 1);

}

int main()

{

srand(time(NULL));

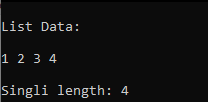
int num = rand()%100;

cout<<num;

guess(num, 0);

}

Task3:



#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int d) : data(d), next(NULL) {}

};

// struct Node

// {

// int data;

// Node \*next;

// };

class Singli

{

public:

Node \*head, \*tail;

Singli()

{

head = NULL;

tail = NULL;

}

void insert\_at\_tail(int num)

{

Node \*temp = new Node(num);

if (head == NULL)

{

head = temp;

tail = temp;

}

else

{

tail->next = temp;

tail = tail->next;

tail->next = NULL;

}

}

void display()

{

Node \*temp = head;

cout << "\nList Data: \n\n";

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

}

int singli\_length(Node \*temp)

{

int static count = 0;

if (temp == NULL)

return count;

count++;

singli\_length(temp->next);

}

};

int main()

{

Singli obj;

obj.insert\_at\_tail(1);

obj.insert\_at\_tail(2);

obj.insert\_at\_tail(3);

obj.insert\_at\_tail(4);

obj.display();

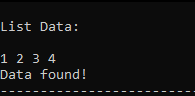
//Singli length

cout<<"\n\nSingli length: "<<obj.singli\_length(obj.head);

return 0;

}

Task4:



#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int d) : data(d), next(NULL) {}

};

// struct Node

// {

// int data;

// Node \*next;

// };

class Singli

{

public:

Node \*head, \*tail;

Singli()

{

head = NULL;

tail = NULL;

}

void insert\_at\_tail(int num)

{

Node \*temp = new Node(num);

if (head == NULL)

{

head = temp;

tail = temp;

}

else

{

tail->next = temp;

tail = tail->next;

tail->next = NULL;

}

}

void display()

{

Node \*temp = head;

cout << "\nList Data: \n\n";

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

}

bool search(Node \*temp, int data)

{

if (temp == NULL)

{

cout<<"\nData Not found!";

return 0;

}

if (temp->data == data)

{

cout<<"\nData found!";

return 1;

}

search(temp->next, data);

}

};

int main()

{

Singli obj;

obj.insert\_at\_tail(1);

obj.insert\_at\_tail(2);

obj.insert\_at\_tail(3);

obj.insert\_at\_tail(4);

obj.display();

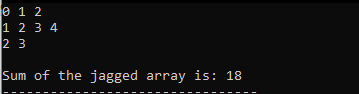
//Search

obj.search(obj.head, 3);

return 0;

}

Task5:



#include <iostream>

using namespace std;

void display\_sum(int \*array[], int rows , int col[]);

void display(int \*array[], int rows , int col[]);

int main()

{

const int rows = 3;

int \*array[rows];

int col[] = {3, 4, 2};

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

{

array[i] = new int[col[i]];

for (int j = 0; j < col[i]; j++)

{

// cout << "\nEnter value for jagged cols: ";

// cin >> array[i][j];

array[i][j] = i+j;

}

}

display(array, rows, col);

display\_sum(array, rows, col);

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

{

delete[] array[i];

}

}

void display\_sum(int \*array[], int rows , int col[])

{

int sum = 0;

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

{

for (int j = 0; j < col[i]; j++)

{

sum += array[i][j];

}

}

cout<<"\nSum of the jagged array is: "<<sum;

}

void display(int \*array[], int rows , int col[])

{

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

{

for (int j = 0; j < col[i]; j++)

{

cout<<array[i][j] <<" ";

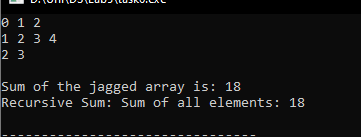
}

cout<<endl;

}

}

Task6:



#include <iostream>

using namespace std;

void display\_sum(int \*array[], int rows , int sizes[]);

void display(int \*array[], int rows , int sizes[]);

void Recursive\_array\_sum(int \*array[], int rows , int sizes[])

{

int static count = 0, sum = 0, i, j;

if (rows < 0)

{

cout << "Sum of all elements: " << sum << endl;

return;

}

i = rows-1;

for (int j = 0; j < sizes[i]; j++)

{

sum += array[i][j];

}

Recursive\_array\_sum(array, i, sizes);

}

int main()

{

const int rows = 3;

int \*array[rows];

int sizes[] = {3, 4, 2};

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

{

array[i] = new int[sizes[i]];

for (int j = 0; j < sizes[i]; j++)

{

// cout << "\nEnter value for jagged sizess: ";

// cin >> array[i][j];

array[i][j] = i+j;

}

}

display(array, rows, sizes);

display\_sum(array, rows, sizes);

cout<<"\nRecursive Sum: ";

Recursive\_array\_sum(array, rows, sizes);

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

{

delete[] array[i];

}

}

void display\_sum(int \*array[], int rows , int sizes[])

{

int sum = 0;

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

{

for (int j = 0; j < sizes[i]; j++)

{

sum += array[i][j];

}

}

cout<<"\nSum of the jagged array is: "<<sum;

}

void display(int \*array[], int rows , int sizes[])

{

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

{

for (int j = 0; j < sizes[i]; j++)

{

cout<<array[i][j] <<" ";

}

cout<<endl;

}

}

Task7:



#include <iostream>

using namespace std;

bool isSafe(int arr[][5], int x, int y, int n)

{

if(x < n && y < n && arr[x][y] == 1)

{

return true;

}

return false;

}

bool lioninMaze(int arr[][5], int x, int y, int n, int solArr[][5] )

{

if((x == (n - 1)) && (y == (n -1 )))

{

solArr[x][y] = 1;

return true;

}

if(isSafe(arr, x, y, n))

{

// solArr[x][y] = 1;

if(lioninMaze(arr, x + 1, y, n, solArr))

{

solArr[x][y] = 1;

return true;

}

if(lioninMaze(arr, x, y + 1, n, solArr))

{

solArr[x][y] = 1;

return true;

}

solArr[x][y] = 0;

return false;

}

return false;

}

int main()

{

int arr[5][5] = {

{1,0,1,0,1},

{1,1,1,1,1},

{0,1,0,1,0},

{1,0,0,1,1},

{1,1,1,0,1}

};

int solArr[5][5] = {0};

lioninMaze(arr, 0, 0, 5, solArr);

// Solution array

for(int i=0; i<5; i++)

{

for(int j=0; j<5; j++)

{

cout<<solArr[i][j]<<" ";

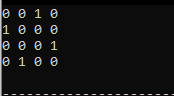
}

cout<<endl;

}

}

Task8:



#include <iostream>

using namespace std;

const int N = 4;

bool isSafe(int board[N][N], int row, int col)

{

int i, j;

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

{

if (board[row][i])

{

return false;

}

}

for (i = row, j=col ; i>=0 && j>=0; i--, j--)

{

if (board[i][j])

{

return false;

}

}

for ( i=row, j=col; i<N && j>=0; i++, j--)

{

if (board[i][j])

{

return false;

}

}

return true;

}

bool placeFlag(int board[N][N], int col)

{

if (col>=N)

{

return true;

}

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

{

if (isSafe(board, i, col))

{

board[i][col] = 1;

if (placeFlag(board, col+1))

{

return true;

}

board[i][col] = 0;

}

}

return false;

}

void solution(int board[N][N])

{

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

{

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

{

cout<<board[i][j]<<" ";

}

cout<<endl;

}

}

bool solve()

{

int board[N][N] = {0};

if (placeFlag(board, 0) == false)

{

cout<<"\nSolution does not exist!";

return false;

}

solution(board);

return true;

}

int main()

{

solve();

return 0;

}