<https://www.youtube.com/watch?v=DneLxrPmmsw&list=PLBlnK6fEyqRi3-lvwLGzcaquOs5OBTCww&index=3>

#include <iostream>

**using** **namespace** std**;**

// Node structure for the linked list

struct Node **{**

int data**;**

Node**\*** next**;**

**};**

// Function to add a node at the end of the linked list

void append**(**Node**\*&** head**,** int data**)** **{**

Node**\*** newNode **=** **new** Node**();**

newNode**->**data **=** data**;**

newNode**->**next **=** **nullptr;**

**if** **(**head **==** **nullptr)** **{**

head **=** newNode**;**

**}** **else** **{**

Node**\*** temp **=** head**;**

**while** **(**temp**->**next **!=** **nullptr)** **{**

temp **=** temp**->**next**;**

**}**

temp**->**next **=** newNode**;**

**}**

**}**

// Function to display the linked list

void display**(**Node**\*** head**)** **{**

Node**\*** temp **=** head**;**

**while** **(**temp **!=** **nullptr)** **{**

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

temp **=** temp**->**next**;**

**}**

cout **<<** "nullptr" **<<** endl**;**

**}**

int main**()** **{**

Node**\*** head **=** **nullptr;**

int choice**,** data**;**

cout **<<** "Linked List Creation Program" **<<** endl**;**

**while** **(true)** **{**

cout **<<** "Enter a number to add to the linked list (or -1 to stop): "**;**

cin **>>** data**;**

**if** **(**data **==** **-**1**)** **break;**

append**(**head**,** data**);**

**}**

cout **<<** "The created linked list is: "**;**

display**(**head**);**

// Clean up memory

Node**\*** temp**;**

**while** **(**head **!=** **nullptr)** **{**

temp **=** head**;**

head **=** head**->**next**;**

**delete** temp**;**

**}**

**return** 0**;**

**}**

// Node structure for the linked list

struct Node **{**

char data**;**

Node**\*** next**;**

Node**(**char val**)** **:** data**(**val**),** next**(nullptr)** **{}**

**};**

// Function to delete a specific node from a singly linked list

void deleteNode**(**Node**\*&** head**,** char key**)** **{**

**if** **(**head **==** **nullptr)** **return;** // List is empty

// If the head node holds the key to be deleted

**if** **(**head**->**data **==** key**)** **{**

Node**\*** temp **=** head**;**

head **=** head**->**next**;**

**delete** temp**;**

**return;**

**}**

// Search for the key to be deleted, keep track of the previous node

Node**\*** current **=** head**;**

Node**\*** prev **=** **nullptr;**

**while** **(**current **!=** **nullptr** **&&** current**->**data **!=** key**)** **{**

prev **=** current**;**

current **=** current**->**next**;**

**}**

// If the key was not found

**if** **(**current **==** **nullptr)** **return;**

// Unlink the node and delete it

prev**->**next **=** current**->**next**;**

**delete** current**;**

**}**

// Function to print the vowels from the linked list

void printVowels**(**Node**\*** head**)** **{**

Node**\*** current **=** head**;**

**while** **(**current **!=** **nullptr)** **{**

char ch **=** current**->**data**;**

**if** **(**ch **==** 'a' **||** ch **==** 'e' **||** ch **==** 'i' **||** ch **==** 'o' **||** ch **==** 'u' **||**

ch **==** 'A' **||** ch **==** 'E' **||** ch **==** 'I' **||** ch **==** 'O' **||** ch **==** 'U'**)** **{**

std**::**cout **<<** ch **<<** " "**;**

**}**

current **=** current**->**next**;**

**}**

std**::**cout **<<** std**::**endl**;**

**}**

// Function to delete the last node of the linked list

void deleteLastNode**(**Node**\*&** head**)** **{**

**if** **(**head **==** **nullptr)** **return;** // List is empty

**if** **(**head**->**next **==** **nullptr)** **{** // Only one node in the list

**delete** head**;**

head **=** **nullptr;**

**return;**

**}**

// Traverse to the second last node

Node**\*** current **=** head**;**

**while** **(**current**->**next**->**next **!=** **nullptr)** **{**

current **=** current**->**next**;**

**}**

**delete** current**->**next**;** // Delete the last node

current**->**next **=** **nullptr;**

**}**

// Function to insert a node with value 50 at the beginning of the linked list

void insertAtBeginning**(**Node**\*&** head**,** int value**)** **{**

Node**\*** newNode **=** **new** Node**(**value**);**

newNode**->**next **=** head**;**

head **=** newNode**;**

**}**

// Helper function to print the linked list

void printList**(**Node**\*** head**)** **{**

Node**\*** current **=** head**;**

**while** **(**current **!=** **nullptr)** **{**

std**::**cout **<<** current**->**data **<<** " -> "**;**

current **=** current**->**next**;**

**}**

std**::**cout **<<** "nullptr" **<<** std**::**endl**;**

**}**

// Function to reverse the linked list

void reverseList**(**Node**\*&** head**)** **{**

Node**\*** prev **=** **nullptr;**

Node**\*** current **=** head**;**

Node**\*** next **=** **nullptr;**

**while** **(**current **!=** **nullptr)** **{**

next **=** current**->**next**;**

current**->**next **=** prev**;**

prev **=** current**;**

current **=** next**;**

**}**

head **=** prev**;**

**}**

// Function to find the middle node of the linked list

Node**\*** findMiddleNode**(**Node**\*** head**)** **{**

**if** **(**head **==** **nullptr)** **return** **nullptr;**

Node**\*** slow **=** head**;**

Node**\*** fast **=** head**;**

**while** **(**fast **!=** **nullptr** **&&** fast**->**next **!=** **nullptr)** **{**

slow **=** slow**->**next**;**

fast **=** fast**->**next**->**next**;**

**}**

**return** slow**;**

**}**

// Function to remove duplicate nodes from a sorted linked list

void removeDuplicates**(**Node**\*&** head**)** **{**

**if** **(**head **==** **nullptr)** **return;**

Node**\*** current **=** head**;**

**while** **(**current**->**next **!=** **nullptr)** **{**

**if** **(**current**->**data **==** current**->**next**->**data**)** **{**

Node**\*** temp **=** current**->**next**;**

current**->**next **=** current**->**next**->**next**;**

**delete** temp**;**

**}** **else** **{**

current **=** current**->**next**;**

**}**

**}**

**}**

int main**()** **{**

// Sample linked list creation for demonstration

Node**\*** head **=** **new** Node**(**'a'**);**

head**->**next **=** **new** Node**(**'b'**);**

head**->**next**->**next **=** **new** Node**(**'c'**);**

head**->**next**->**next**->**next **=** **new** Node**(**'e'**);**

head**->**next**->**next**->**next**->**next **=** **new** Node**(**'i'**);**

std**::**cout **<<** "Original List: "**;**

printList**(**head**);**

// a) Delete a specific node

deleteNode**(**head**,** 'c'**);**

std**::**cout **<<** "After deleting 'c': "**;**

printList**(**head**);**

// c) Print vowels from the linked list

std**::**cout **<<** "Vowels in the list: "**;**

printVowels**(**head**);**

// ii) Delete the last node

deleteLastNode**(**head**);**

std**::**cout **<<** "After deleting the last node: "**;**

printList**(**head**);**

// i) Insert a node with value 50 at the beginning

insertAtBeginning**(**head**,** '50'**);** // Note: '50' is a character for demonstration purposes

std**::**cout **<<** "After inserting '50' at the beginning: "**;**

printList**(**head**);**

// Reverse the linked list

reverseList**(**head**);**

std**::**cout **<<** "After reversing the list: "**;**

printList**(**head**);**

// Find the middle node

Node**\*** middle **=** findMiddleNode**(**head**);**

**if** **(**middle **!=** **nullptr)** **{**

std**::**cout **<<** "Middle node: " **<<** middle**->**data **<<** std**::**endl**;**

**}**

// Remove duplicates from a sorted linked list

insertAtBeginning**(**head**,** 'e'**);** // Add duplicate for demonstration

insertAtBeginning**(**head**,** 'e'**);**

removeDuplicates**(**head**);**

std**::**cout **<<** "After removing duplicates: "**;**

printList**(**head**);**

**return** 0**;**

**}**

-----------------------------------------------------------------------------  
// Node structure for the linked list

struct Node **{**

char data**;**

Node**\*** next**;**

Node**(**char val**)** **:** data**(**val**),** next**(nullptr)** **{}**

**};**

// Function to delete a specific node from a singly linked list

void deleteNode**(**Node**\*&** head**,** char key**)** **{**

**if** **(**head **==** **nullptr)** **return;** // List is empty

// If the head node holds the key to be deleted

**if** **(**head**->**data **==** key**)** **{**

Node**\*** temp **=** head**;**

head **=** head**->**next**;**

**delete** temp**;**

**return;**

**}**

// Search for the key to be deleted, keep track of the previous node

Node**\*** current **=** head**;**

Node**\*** prev **=** **nullptr;**

**while** **(**current **!=** **nullptr** **&&** current**->**data **!=** key**)** **{**

prev **=** current**;**

current **=** current**->**next**;**

**}**

// If the key was not found

**if** **(**current **==** **nullptr)** **return;**

// Unlink the node and delete it

prev**->**next **=** current**->**next**;**

**delete** current**;**

**}**

// Function to print the vowels from the linked list

void printVowels**(**Node**\*** head**)** **{**

Node**\*** current **=** head**;**

**while** **(**current **!=** **nullptr)** **{**

char ch **=** current**->**data**;**

**if** **(**ch **==** 'a' **||** ch **==** 'e' **||** ch **==** 'i' **||** ch **==** 'o' **||** ch **==** 'u' **||**

ch **==** 'A' **||** ch **==** 'E' **||** ch **==** 'I' **||** ch **==** 'O' **||** ch **==** 'U'**)** **{**

std**::**cout **<<** ch **<<** " "**;**

**}**

current **=** current**->**next**;**

**}**

std**::**cout **<<** std**::**endl**;**

**}**

// Function to delete the last node of the linked list

void deleteLastNode**(**Node**\*&** head**)** **{**

**if** **(**head **==** **nullptr)** **return;** // List is empty

**if** **(**head**->**next **==** **nullptr)** **{** // Only one node in the list

**delete** head**;**

head **=** **nullptr;**

**return;**

**}**

// Traverse to the second last node

Node**\*** current **=** head**;**

**while** **(**current**->**next**->**next **!=** **nullptr)** **{**

current **=** current**->**next**;**

**}**

**delete** current**->**next**;** // Delete the last node

current**->**next **=** **nullptr;**

**}**

// Function to insert a node with value 50 at the beginning of the linked list

void insertAtBeginning**(**Node**\*&** head**,** int value**)** **{**

Node**\*** newNode **=** **new** Node**(**value**);**

newNode**->**next **=** head**;**

head **=** newNode**;**

**}**

// Helper function to print the linked list

void printList**(**Node**\*** head**)** **{**

Node**\*** current **=** head**;**

**while** **(**current **!=** **nullptr)** **{**

std**::**cout **<<** current**->**data **<<** " -> "**;**

current **=** current**->**next**;**

**}**

std**::**cout **<<** "nullptr" **<<** std**::**endl**;**

**}**

int main**()** **{**

// Sample linked list creation for demonstration

Node**\*** head **=** **new** Node**(**'a'**);**

head**->**next **=** **new** Node**(**'b'**);**

head**->**next**->**next **=** **new** Node**(**'c'**);**

head**->**next**->**next**->**next **=** **new** Node**(**'e'**);**

head**->**next**->**next**->**next**->**next **=** **new** Node**(**'i'**);**

std**::**cout **<<** "Original List: "**;**

printList**(**head**);**

// a) Delete a specific node

deleteNode**(**head**,** 'c'**);**

std**::**cout **<<** "After deleting 'c': "**;**

printList**(**head**);**

// c) Print vowels from the linked list

std**::**cout **<<** "Vowels in the list: "**;**

printVowels**(**head**);**

// ii) Delete the last node

deleteLastNode**(**head**);**

std**::**cout **<<** "After deleting the last node: "**;**

printList**(**head**);**

// i) Insert a node with value 50 at the beginning

insertAtBeginning**(**head**,** '50'**);** // Note: '50' is a character for demonstration purposes

std**::**cout **<<** "After inserting '50' at the beginning: "**;**

printList**(**head**);**

**return** 0**;**

**}**

#include <iostream>

#include <string>

using namespace std**;**

// Node structure for the linked list

struct Node **{**

int id**;**

string name**;**

float price**;**

int stock\_quantity**;**

string category**;**

Node**\*** next**;**

**};**

// Function to insert a new product node into the linked list

void insert**(**Node**\*&** head**,** int id**,** string name**,** float price**,** int stock\_quantity**,** string category**)** **{**

Node**\*** newNode **=** new Node**{**id**,** name**,** price**,** stock\_quantity**,** category**,** nullptr**};**

**if** **(!**head**)** **{**

head **=** newNode**;**

**return;**

**}**

Node**\*** temp **=** head**;**

**while** **(**temp**->**next**)** **{**

temp **=** temp**->**next**;**

**}**

temp**->**next **=** newNode**;**

**}**

// Function to calculate price difference

float priceDifference**(**Node**\*** head**)** **{**

**if** **(!**head**)** **return** 0**;**

float maxPrice **=** head**->**price**,** minPrice **=** head**->**price**;**

Node**\*** temp **=** head**;**

**while** **(**temp**)** **{**

**if** **(**temp**->**price **>** maxPrice**)** maxPrice **=** temp**->**price**;**

**if** **(**temp**->**price **<** minPrice**)** minPrice **=** temp**->**price**;**

temp **=** temp**->**next**;**

**}**

**return** maxPrice **-** minPrice**;**

**}**

// Function to calculate total stock value

float totalStockValue**(**Node**\*** head**)** **{**

float totalValue **=** 0**;**

Node**\*** temp **=** head**;**

**while** **(**temp**)** **{**

totalValue **+=** temp**->**price **\*** temp**->**stock\_quantity**;**

temp **=** temp**->**next**;**

**}**

**return** totalValue**;**

**}**

// Function to find and print products with price <= 50

void findAffordableProducts**(**Node**\*** head**)** **{**

Node**\*** temp **=** head**;**

cout **<<** "Products with price <= 50: "**;**

**while** **(**temp**)** **{**

**if** **(**temp**->**price **<=** 50**)** **{**

cout **<<** temp**->**name **<<** " "**;**

**}**

temp **=** temp**->**next**;**

**}**

cout **<<** endl**;**

**}**

// Function to display product details

void displayProducts**(**Node**\*** head**)** **{**

Node**\*** temp **=** head**;**

**while** **(**temp**)** **{**

cout **<<** "ID: " **<<** temp**->**id **<<** ", Name: " **<<** temp**->**name **<<** ", Price: $" **<<** temp**->**price

**<<** ", Stock: " **<<** temp**->**stock\_quantity **<<** ", Category: " **<<** temp**->**category **<<** endl**;**

temp **=** temp**->**next**;**

**}**

**}**

int main**()** **{**

Node**\*** head **=** nullptr**;**

// Insert products into the linked list

insert**(**head**,** 101**,** "Laptop"**,** 799.99**,** 10**,** "Electronics"**);**

insert**(**head**,** 102**,** "Notebook"**,** 2.99**,** 100**,** "Stationery"**);**

insert**(**head**,** 103**,** "Pen"**,** 1.50**,** 200**,** "Stationery"**);**

insert**(**head**,** 104**,** "Headphones"**,** 49.99**,** 25**,** "Electronics"**);**

// Display all product records

cout **<<** "Product Records:" **<<** endl**;**

displayProducts**(**head**);**

// Calculate and display price difference

cout **<<** "Price difference between the most expensive and the cheapest product: $" **<<** priceDifference**(**head**)** **<<** endl**;**

// Calculate and display total stock value

cout **<<** "Total stock value of all products: $" **<<** totalStockValue**(**head**)** **<<** endl**;**

// Find and display affordable products

findAffordableProducts**(**head**);**

**return** 0**;**

**}**

#include <iostream>

#include <string>

using namespace std**;**

// Structure to hold product data

struct Product **{**

int id**;**

string name**;**

float price**;**

int stock\_quantity**;**

string category**;**

**};**

// Function to calculate price difference

float priceDifference**(**Product products**[],** int size**)** **{**

float maxPrice **=** products**[**0**].**price**,** minPrice **=** products**[**0**].**price**;**

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

**if** **(**products**[**i**].**price **>** maxPrice**)** maxPrice **=** products**[**i**].**price**;**

**if** **(**products**[**i**].**price **<** minPrice**)** minPrice **=** products**[**i**].**price**;**

**}**

**return** maxPrice **-** minPrice**;**

**}**

// Function to calculate total stock value

float totalStockValue**(**Product products**[],** int size**)** **{**

float totalValue **=** 0**;**

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

totalValue **+=** products**[**i**].**price **\*** products**[**i**].**stock\_quantity**;**

**}**

**return** totalValue**;**

**}**

// Function to find products with price <= 50

void findAffordableProducts**(**Product products**[],** int size**)** **{**

cout **<<** "Products with price <= 50: "**;**

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

**if** **(**products**[**i**].**price **<=** 50**)** **{**

cout **<<** products**[**i**].**name **<<** " "**;**

**}**

**}**

cout **<<** endl**;**

**}**

int main**()** **{**

// Array of products

Product products**[**4**]** **=** **{**

**{**101**,** "Laptop"**,** 799.99**,** 10**,** "Electronics"**},**

**{**102**,** "Notebook"**,** 2.99**,** 100**,** "Stationery"**},**

**{**103**,** "Pen"**,** 1.50**,** 200**,** "Stationery"**},**

**{**104**,** "Headphones"**,** 49.99**,** 25**,** "Electronics"**}**

**};**

cout **<<** "Price difference: " **<<** priceDifference**(**products**,** 4**)** **<<** endl**;**

cout **<<** "Total stock value: " **<<** totalStockValue**(**products**,** 4**)** **<<** endl**;**

findAffordableProducts**(**products**,** 4**);**

**return** 0**;**

**}**