Student Name: Sahul Kumar Parida **UID: 20BCS4919**
Branch: CSE **Section/Group: G-79**

Question 1

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters. Letters are case sensitive.

Code:

```
#include <iostream>
#include <string>

using namespace std;

int longestPalindrome(string str) {
  // Create a table to store the longest palindromic substrings.
  bool dp[str.length()][str.length()];
  for (int i = 0; i < str.length(); i++) {
    for (int j = 0; j < str.length(); j++) {
      dp[i][j] = false;
    }
  }

  // Fill the table in the bottom-up manner.
  for (int i = 0; i < str.length(); i++) {
    for (int j = 0; j <= i; j++) {
      if (str[i] == str[j] && (i - j <= 1 || dp[j + 1][i - 1])) {
        dp[i][j] = true;
      }
    }
  }

  // Find the length of the longest palindrome.
  int maxLen = 0;
  for (int i = 0; i < str.length(); i++) {
    for (int j = 0; j <= i; j++) {
      if (dp[i][j] && maxLen < (i - j + 1)) {
        maxLen = (i - j + 1);
      }
    }
  }

  return maxLen;
}
```

```
int main() {
  string str = "madam";
  int maxLen = longestPalindrome(str);
  cout << "The length of the longest palindrome is " << maxLen << endl;
  return 0;
}
```

Question 2

A data structure needs to be implemented in such a way that we have the references i.e. the addresses of the values. None of the addresses are in a continuous memory block. Each time a new value needs to be stored, we need to allocate memory. Write a program to implement the following:

1. Addition of a new value at a given position
2. Print all the values in the list
3. Delete a given value from a location

Code:

```
#include <iostream>

using namespace std;

struct Node {
  int value;
  Node* next;
};

Node* head = nullptr;

void addNode(int value, int position) {
  // Create a new node.
  Node* newNode = new Node();
  newNode->value = value;
  newNode->next = nullptr;

  // If the list is empty, make the new node the head.
  if (head == nullptr) {
    head = newNode;
    return;
  }

  // Otherwise, find the node at the given position.
  Node* currentNode = head;
  for (int i = 0; i < position - 1; i++) {
```

```
    currentNode = currentNode->next;
    if (currentNode == nullptr) {
      cout << "The position is out of bounds." << endl;
      return;
    }
  }

  // Insert the new node after the current node.
  newNode->next = currentNode->next;
  currentNode->next = newNode;
}

void printList() {
  // Initialize a pointer to the head of the list.
  Node* currentNode = head;

  // Iterate over the list and print the values.
  while (currentNode != nullptr) {
    cout << currentNode->value << " ";
    currentNode = currentNode->next;
  }

  cout << endl;
}

void deleteNode(int position) {
  // If the list is empty, do nothing.
  if (head == nullptr) {
    return;
  }

  // Otherwise, find the node at the given position.
  Node* currentNode = head;
  for (int i = 0; i < position - 1; i++) {
    currentNode = currentNode->next;
    if (currentNode == nullptr) {
      cout << "The position is out of bounds." << endl;
      return;
    }
  }

  // If the node at the given position is the head, make the next node the head.
  if (currentNode->next == nullptr) {
    head = nullptr;
    return;
  }
```

```
  // Otherwise, set the next node of the previous node to the next node of the given node.
  currentNode->next = currentNode->next->next;
}

int main() {
  // Add some nodes to the list.
  addNode(10, 0);
  addNode(20, 1);
  addNode(30, 2);

  // Print the list.
  printList();

  // Delete a node from the list.
  deleteNode(1);

  // Print the list again.
  printList();

  return 0;
}
```

Question 3

Characters are sorted in a form of a list (non continuous). We have to check if the list of characters has a given character in it or not.

Code:

```
#include <iostream>
#include <vector>

using namespace std;

bool isCharacterPresent(vector<char> characters, char character) {
  // Initialize the low and high pointers.
  int low = 0;
  int high = characters.size() - 1;

  // Iterate until the low pointer is greater than or equal to the high pointer.
  while (low <= high) {
    // Calculate the middle index.
    int mid = (low + high) / 2;
```

```
    // If the character is found, return true.
    if (characters[mid] == character) {
      return true;
    } else if (characters[mid] < character) {
      // Set the low pointer to the middle index + 1.
      low = mid + 1;
    } else {
      // Set the high pointer to the middle index - 1.
      high = mid - 1;
    }
  }

  // Return false if the character is not found.
  return false;
}

int main() {
  // Create a list of characters.
  vector<char> characters = {'a', 'b', 'c', 'd', 'e'};

  // Check if the character 'c' is present in the list.
  bool isPresent = isCharacterPresent(characters, 'c');

  // Print the result.
  if (isPresent) {
    cout << "The character 'c' is present in the list." << endl;
  } else {
    cout << "The character 'c' is not present in the list." << endl;
  }

  return 0;
}
```

Question 4

Write a program in C++ to print common elements in 3 sorted arrays

Code:

```
#include <iostream>
#include <vector>

using namespace std;

void printCommonElements(vector<int> arr1, vector<int> arr2, vector<int> arr3) {
  // Initialize two pointers to the start of each array.
```

```
  int i = 0;
  int j = 0;
  int k = 0;

  // Iterate until all three pointers have reached the end of their respective arrays.
  while (i < arr1.size() && j < arr2.size() && k < arr3.size()) {
    // If the current elements of the three arrays are equal, print them and increment the
pointer with the largest element.
    if (arr1[i] == arr2[j] == arr3[k]) {
      cout << arr1[i] << " ";
      i++;
      j++;
      k++;
    } else if (arr1[i] < arr2[j]) {
      // Increment the pointer of arr1 as it has the smallest element.
      i++;
    } else if (arr2[j] < arr3[k]) {
      // Increment the pointer of arr2 as it has the smallest element.
      j++;
    } else {
      // Increment the pointer of arr3 as it has the smallest element.
      k++;
    }
  }
}

int main() {
  // Create three sorted arrays.
  vector<int> arr1 = {1, 2, 3, 4, 5};
  vector<int> arr2 = {2, 3, 4, 5, 6};
  vector<int> arr3 = {3, 4, 5, 6, 7};

  // Print the common elements in the three arrays.
  printCommonElements(arr1, arr2, arr3);

  return 0;
}
```

Question 5

You are given a linked list that contains N integers. You have performed the following reverse operation on the list:

- Select all the subparts of the list that contain only even integers. For example, if the list is {1,2,8,9,12,16}, then the selected subparts will be {2,8}, {12,16}.
- Reverse the selected subpart such as {8,2} and {16,12}.

Now, you are required to retrieve the original list.
Note: You should use the following definition of the linked list for this problem:

```
class Node {
   Object data;
   Node next;
}
```

Code:

```
#include <iostream>
#include <vector>

using namespace std;

class Node {
  public:
    int data;
    Node* next;

    Node(int data) {
      this->data = data;
      this->next = nullptr;
    }
};

void reverseSublist(Node* head, int start, int end) {
  // Initialize two pointers to the start and end of the sublist.
  Node* current = head;
  Node* previous = nullptr;

  // Iterate through the sublist and reverse the nodes.
  for (int i = start; i <= end; i++) {
    Node* next = current->next;
    current->next = previous;
    previous = current;
    current = next;
  }
```

```
  // Update the next pointer of the previous node.
  if (previous != nullptr) {
    previous->next = current;
  } else {
    head = current;
  }
}

void retrieveOriginalList(Node* head) {
  // Initialize a vector to store the original list.
  vector<int> originalList;

  // Iterate through the linked list and add the elements to the vector.
  Node* current = head;
  while (current != nullptr) {
    originalList.push_back(current->data);
    current = current->next;
  }

  // Reverse the vector.
  reverse(originalList.begin(), originalList.end());

  // Print the original list.
  for (int i = 0; i < originalList.size(); i++) {
    cout << originalList[i] << " ";
  }
}

int main() {
  // Create a linked list.
  Node* head = new Node(1);
  head->next = new Node(2);
  head->next->next = new Node(8);
  head->next->next->next = new Node(9);
  head->next->next->next->next = new Node(12);
  head->next->next->next->next->next = new Node(16);

  // Reverse the subparts of the linked list.
  reverseSublist(head, 1, 3);
  reverseSublist(head, 4, 5);

  // Retrieve the original list.
  retrieveOriginalList(head);

  return 0;
```

}

Question 6

You have a matrix S consisting of N rows and M columns. Let u be the maximum element of the matrix and v be the smallest element of the matrix. If any element whose value is equal to u or v are called unsafe elements and they disfigure the complete row and column of the matrix. More formally, if any element is equal to u or v and contains cell number (x, y), that is, S[x][y]=u or S[x][y]=v is unsafe so that they also disfigure all the cells that have row x or column y and also are unsafe.

Your task is to find the number of safe elements.

Code:

```
#include <iostream>
#include <vector>

using namespace std;

int findNumberOfSafeElements(vector<vector<int>>& matrix, int N, int M, int u, int
v) {
  // Initialize a vector to store the unsafe rows.
  vector<bool> unsafeRows(N, false);

  // Initialize a vector to store the unsafe columns.
  vector<bool> unsafeColumns(M, false);

  // Iterate through the matrix and mark the unsafe rows and columns.
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
      if (matrix[i][j] == u || matrix[i][j] == v) {
        unsafeRows[i] = true;
        unsafeColumns[j] = true;
      }
    }
  }

  // Initialize a counter to store the number of safe elements.
  int count = 0;

  // Iterate through the matrix and count the number of safe elements.
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
      if (!unsafeRows[i] && !unsafeColumns[j]) {
        count++;
      }
```

```
    }
  }

  // Return the number of safe elements.
  return count;
}

int main() {
  // Create a matrix.
  vector<vector<int>> matrix = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12},
    {13, 14, 15, 16}
  };

  // Initialize the maximum and minimum elements of the matrix.
  int u = 16;
  int v = 1;

  // Find the number of safe elements.
  int count = findNumberOfSafeElements(matrix, 4, 4, u, v);

  // Print the number of safe elements.
  cout << count << endl;

  return 0;
}
```