Linked List:

class Node

{

public:

int data;

Node\* next;

Node()

{

data = 0;

next = NULL;

}

Node(int x)

{

data = x;

next = NULL;

}

void setData(int x)

{

data = x;

}

void setNext(Node\* ptr)

{

next = ptr;

}

int getData()

{

return data;

}

Node\* getNext()

{

return next;

}

};

class SLinkedList

{

public:

Node\* head;

Node\* current;

SLinkedList()

{

head = NULL;

current = NULL;

}

void insert(int x)

{

Node\* ptr = new Node;

ptr->data = x;

if (head == NULL)

{

head = ptr;

return;

}

current = head;

while (current->next != NULL)

{

current = current->next;

}

current->next = ptr;

}

void insertAtHead(int x)

{

Node\* ptr = new Node;

ptr->data = x;

ptr->next = head;

head = ptr;

}

bool isEmpty()

{

if (head == NULL)

return 1;

return 0;

}

//Q5: isFull() is not needed as list has no fixed size or capacity and new element can always be added

int search(int x)

{

current = head;

int count = 0;

while (current != NULL)

{

if (current->data == x)

return count;

current = current->next;

count++;

}

return -1;

}

void update(int val1, int val2)

{

current = head;

while (current != NULL)

{

if (current->data == val1)

{

current->data = val2;

return;

}

current = current->next;

}

}

void InsertAtIndex(int x, int idx)

{

Node\* ptr = new Node;

ptr->data = x;

if (idx < 0)

{

return;

}

if (idx == 0)

{

ptr->next = head;

head = ptr;

}

int count = 0;

current = head;

while (current != NULL && count < idx)

{

count++;

current = current->next;

}

if (!current)

{

return;

}

ptr->next = current->next;

current->next = ptr;

}

void remove(int x)

{

Node\* previous = NULL;

current = head;

int count = 0;

while (current != NULL)

{

if (current->data == x)

break;

previous = current;

current = current->next;

count++;

}

if (current != NULL && previous == NULL)

{

head = current->next;

delete current;

}

else if (current != NULL && previous != NULL)

{

previous->next = current->next;

delete current;

}

}

void print()

{

current = head;

while (current != NULL)

{

cout << current->data << endl;

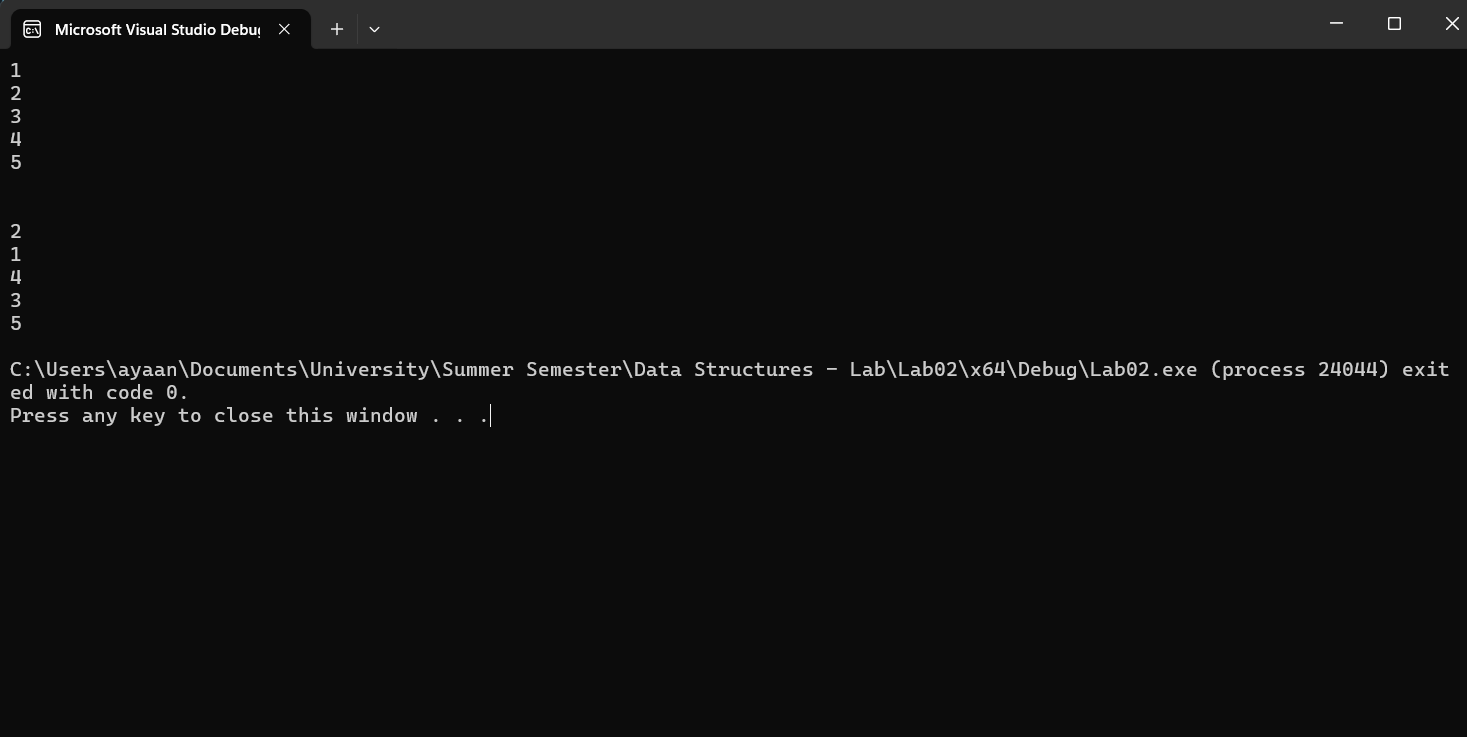
current = current->next;

}

}

};

Task 01:



Node\* swapAdjacentNodes(Node\* head)

{

if (!head || !head->next)

{

return head;

}

Node\* newHead = head->next;

Node\* current = head;

Node\* previous = NULL;

while (current && current->next)

{

Node\* next = current->next;

Node\* temp = next->next;

next->next = current;

current->next = temp;

if (previous)

{

previous->next = next;

}

previous = current;

current = temp;

}

cout << endl;

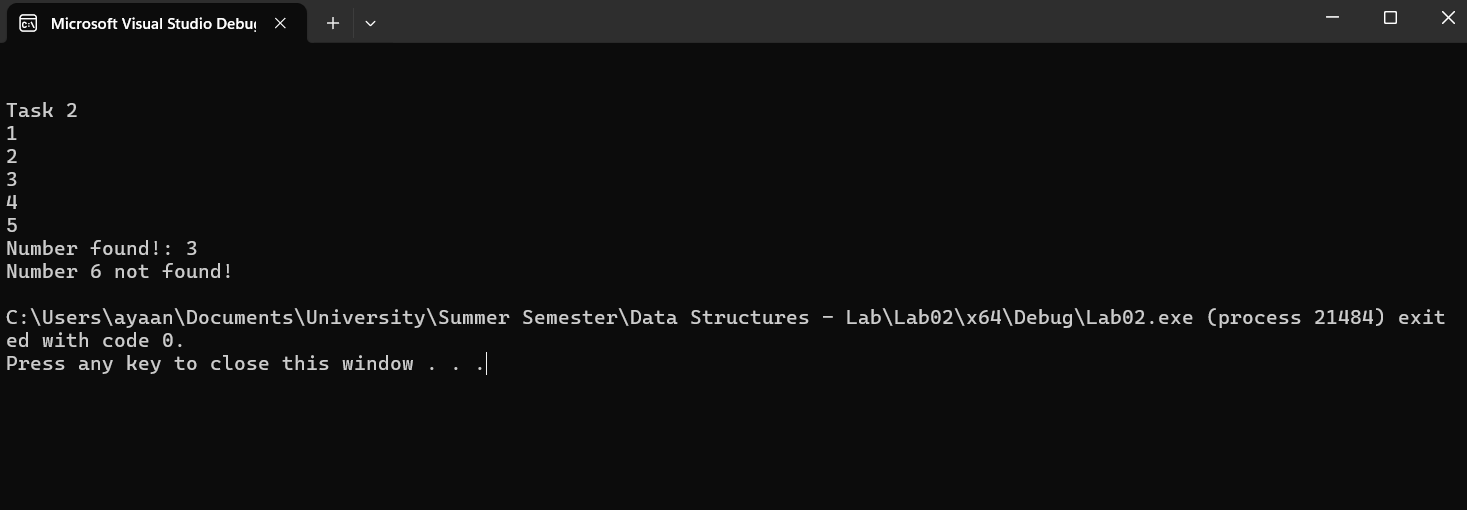
head = newHead;

//cout << head->data;

return newHead;

}

Task 02:



void search(int x, Node\* head)

{

Node\* current = new Node;

current = head;

//int count = 0;

bool flag = 0;

while (current != NULL)

{

if (current->data == x)

flag = 1;

current = current->next;

//count++;

}

if (!flag)

{

cout<<"Number "<< x << " not found!\n";

}

else

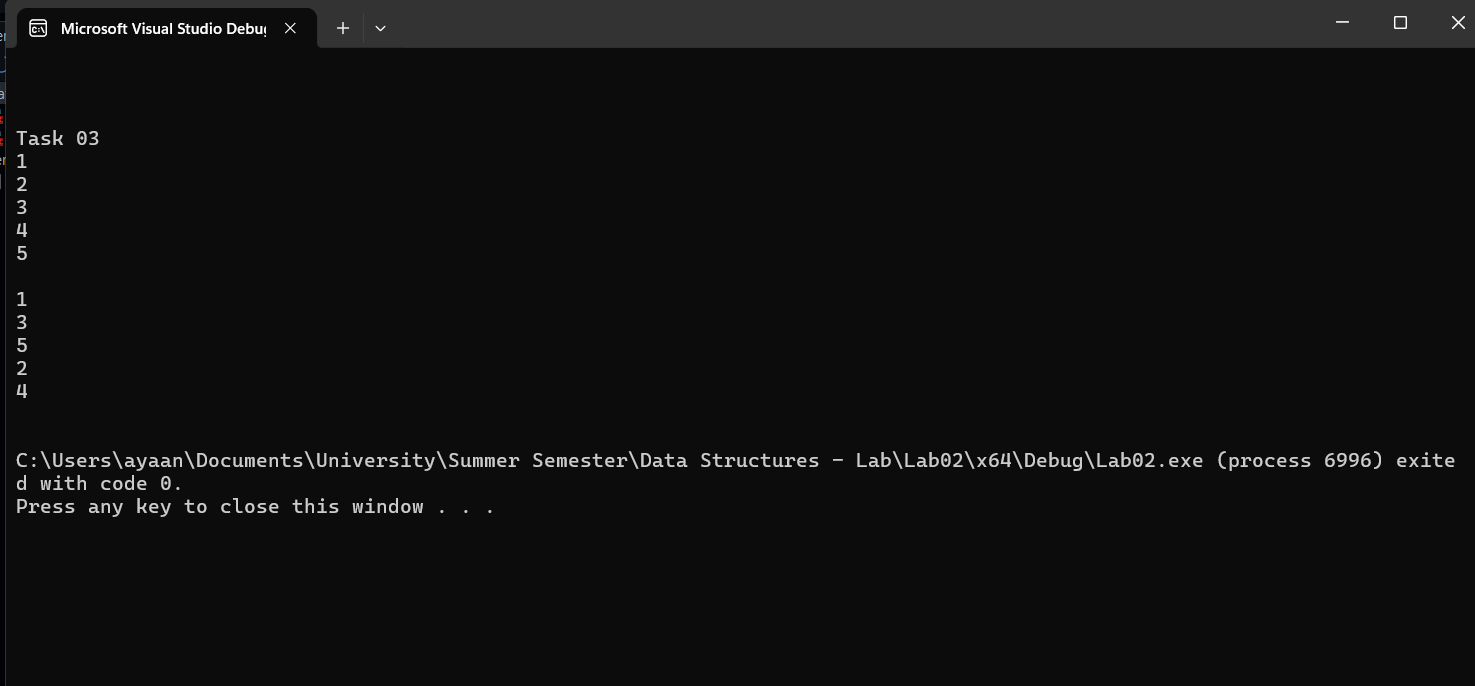
{

cout << "Number found!: " << x << endl;

}

}

Q3:



SLinkedList groupOdds(SLinkedList& list3)

{

Node\* current = new Node;

Node\* prev = new Node;

current = list3.head;

prev = list3.head;

SLinkedList odd;

SLinkedList even;

SLinkedList final;

while (current != NULL)

{

if (current->data % 2 == 1)

{

odd.insert(current->data);

}

else

{

even.insert(current->data);

}

current = current->next;

}

current = odd.head;

while (current != NULL)

{

final.insert(current->data);

current = current->next;

}

current = even.head;

while (current != NULL)

{

final.insert(current->data);

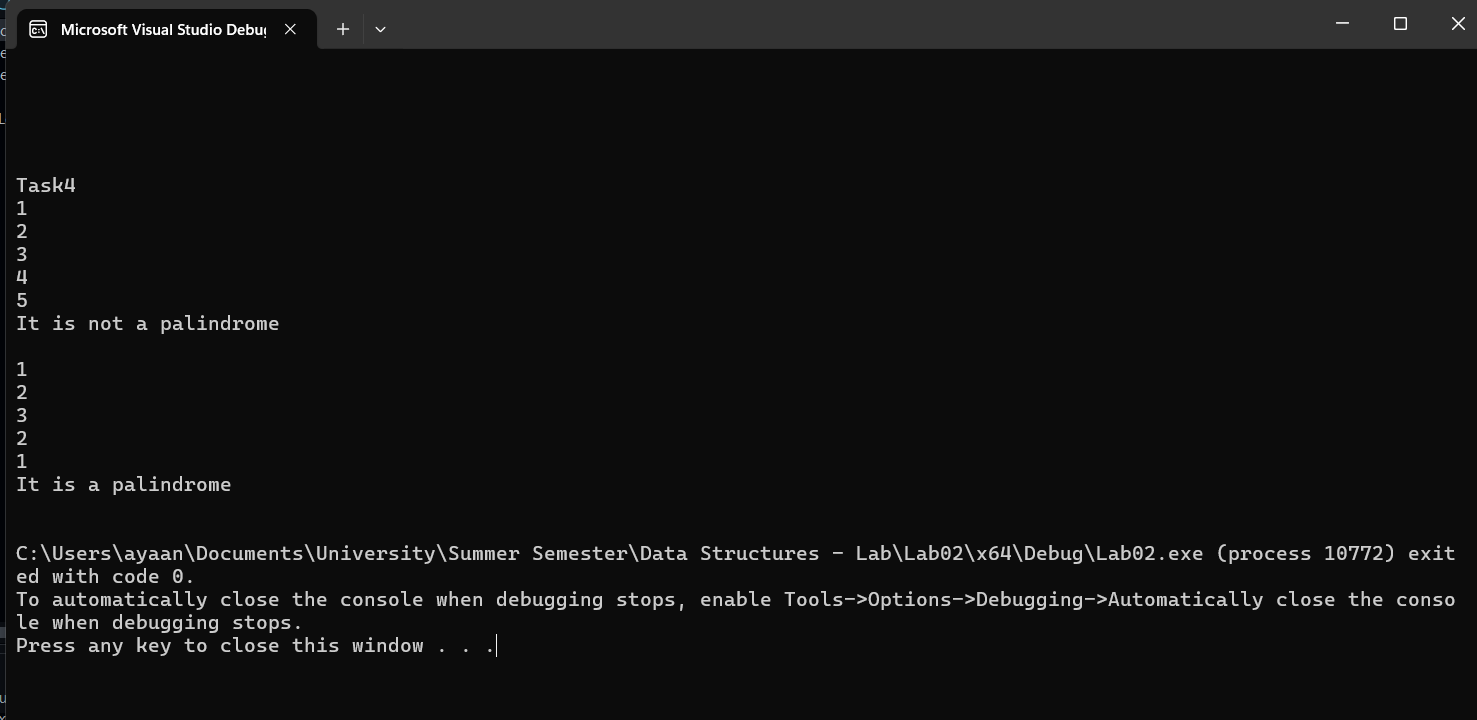
current = current->next;

}

return final;

}

Q4:



void checkPalindrome(SLinkedList list4)

{

Node\* start = list4.head;

Node\* current = list4.head;

Node\* end;

SLinkedList temp;

while (current != NULL)

{

temp.insertAtHead(current->data);

current = current->next;

}

bool flag = 1;

current = list4.head;

end = temp.head;

//cout << "temp";

//temp.print();

while (current->data == end->data)

{

int val1 = current->data;

int val2 = end->data;

if (current->next == NULL || end->next == NULL)

{

cout << "It is a palindrome\n";

return;

}

current = current->next;

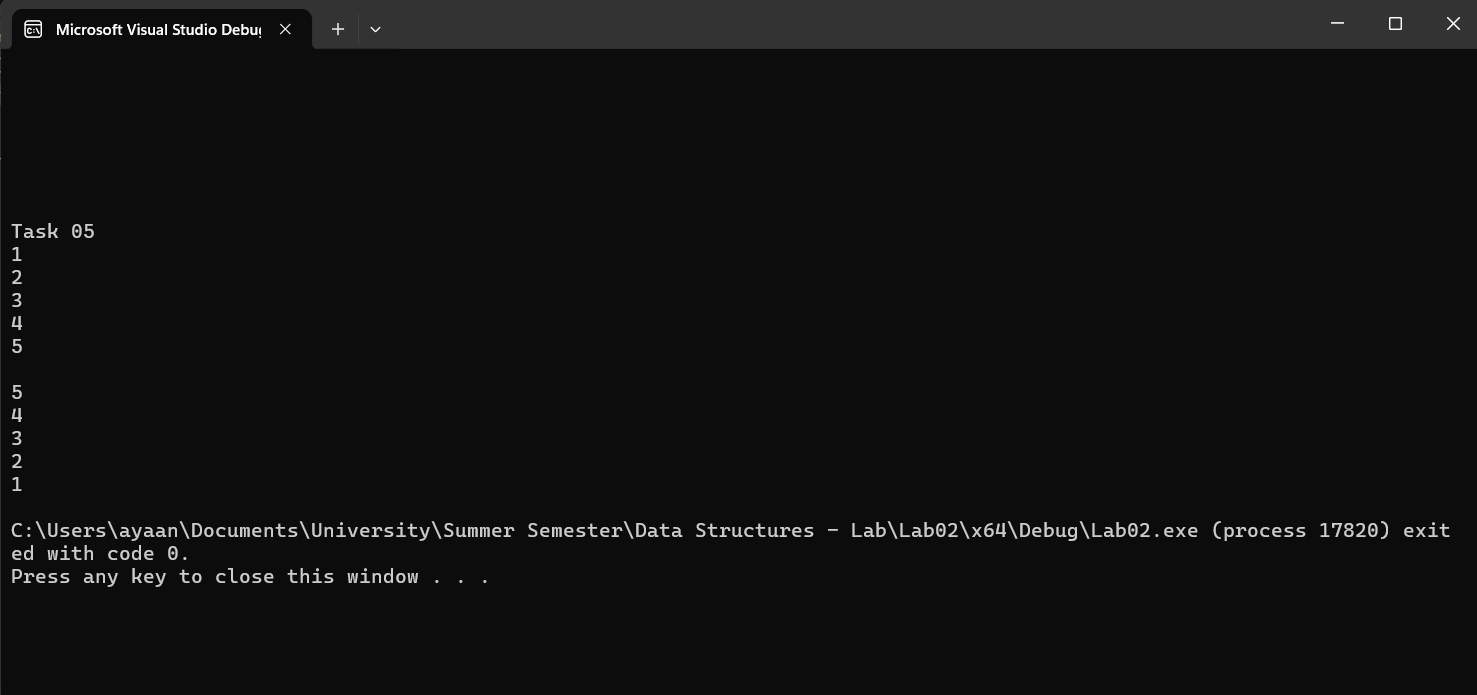
end = end->next;

}

cout << "It is not a palindrome\n";

}

Q5:



void reverseList(SLinkedList& list)

{

Node\* start = list.head;

Node\* current = list.head;

Node\* end;

SLinkedList temp;

while (current != NULL)

{

temp.insertAtHead(current->data);

current = current->next;

}

current = list.head;

end = temp.head;

while (current != NULL)

{

current->data = end->data;

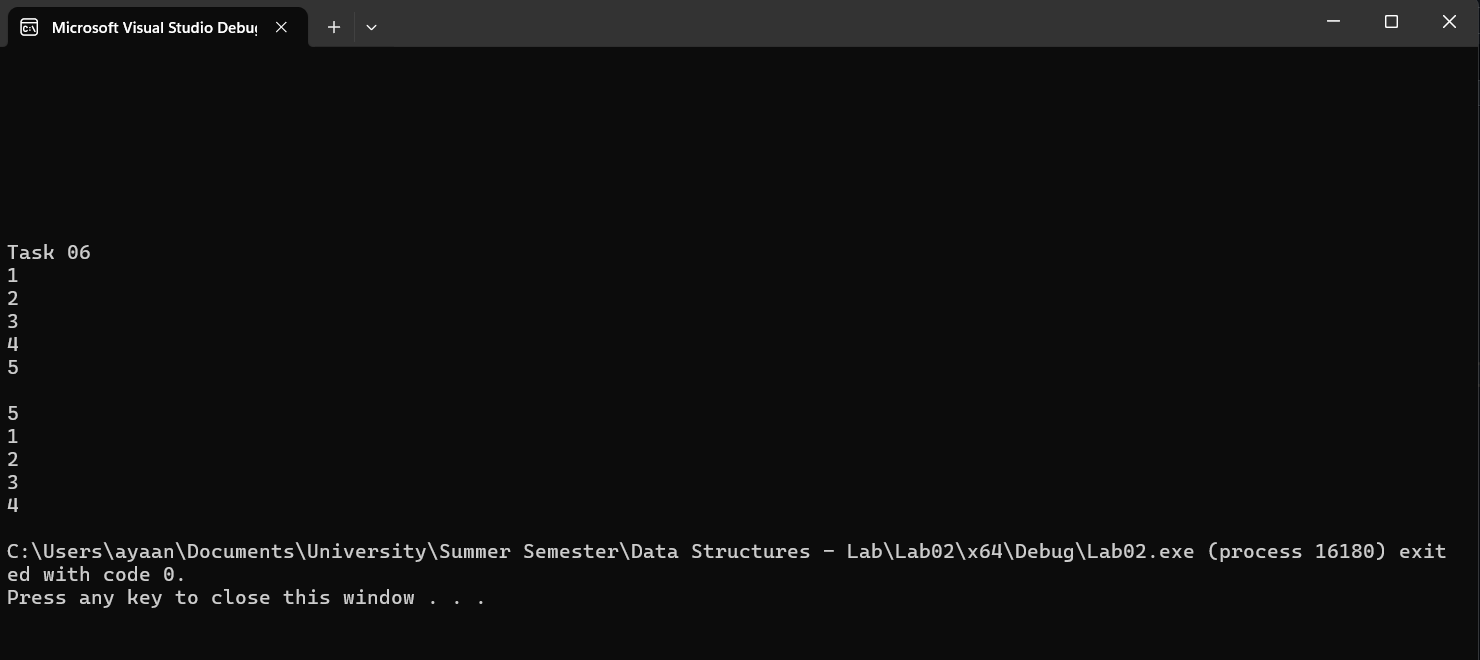
current = current->next;

end = end->next;

}

}

Task 06:



void moveTailtoHead(SLinkedList& list)

{

Node\* current = list.head;

Node\* end;

while (current->next->next != NULL)

{

current = current->next;

}

end = current->next;

current->next = NULL;

list.insertAtHead(end->data);

}