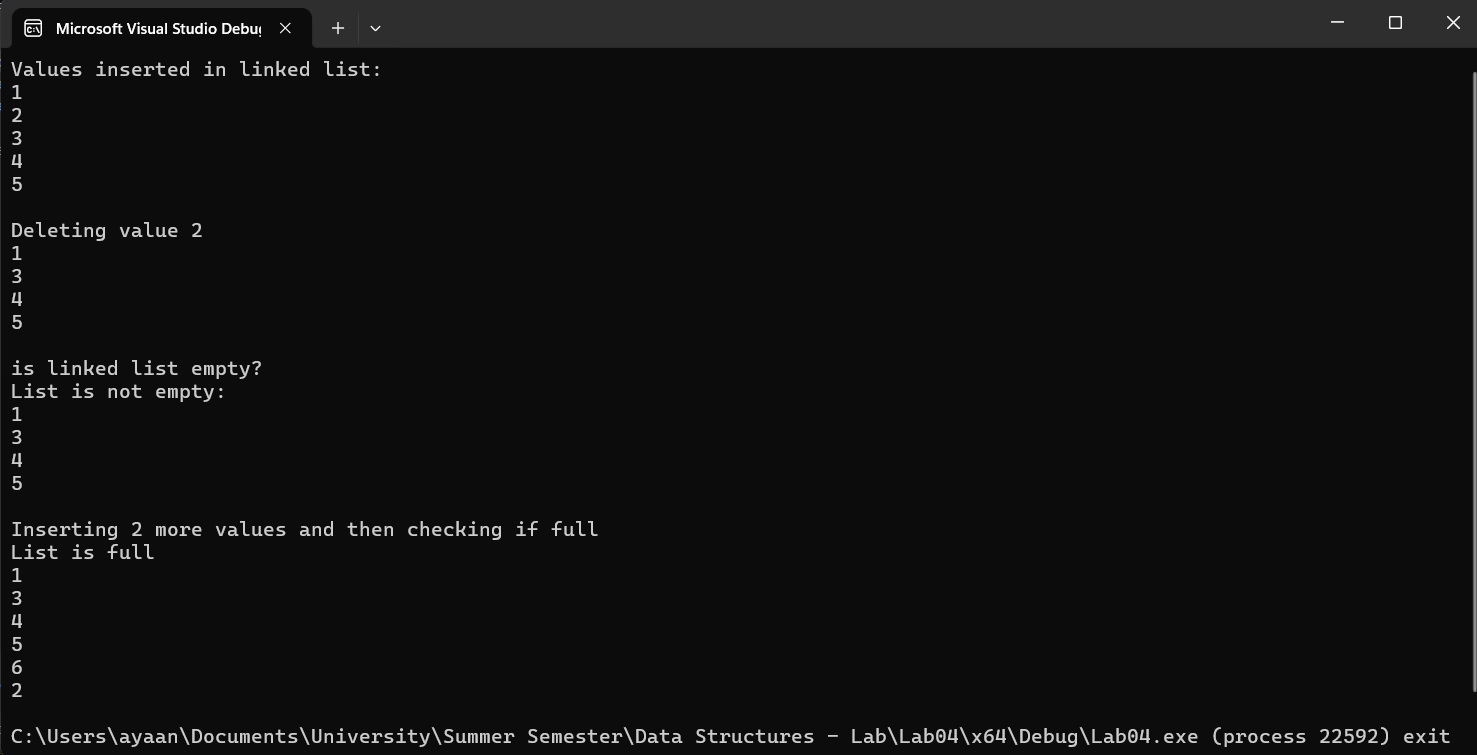
Q1:



#pragma once

#include <iostream>

using namespace std;

class DoublyNode

{

public:

int data;

DoublyNode\* prev;

DoublyNode\* next;

DoublyNode()

{

data = 0;

prev = NULL;

next = NULL;

}

DoublyNode(int x1, DoublyNode\* x2, DoublyNode\* x3)

{

data = x1;

prev = x2;

next = x3;

}

void setData(int x1)

{

data = x1;

}

void setPrev(DoublyNode\* x1)

{

prev = x1;

}

void setNext(DoublyNode\* x1)

{

next = x1;

}

int getData()

{

return data;

}

DoublyNode\* getPrev()

{

return prev;

}

DoublyNode\* getNext()

{

return next;

}

};

class DoublyLinkedList

{

public:

DoublyNode\* head;

DoublyNode\* current;

int size;

DoublyLinkedList()

{

head = NULL;

current = NULL;

size = 6;

}

DoublyNode\* getHead()

{

return head;

}

void insert(int x)

{

DoublyNode\* ptr = new DoublyNode;

ptr->setData(x);

if (head == NULL)

{

head = ptr;

return;

}

current = head;

int count = 1;

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

{

current = current->getNext();

count++;

}

if (count >= size)

{

cout << "List is full can't insert value\n";

return;

}

current->setNext(ptr);

ptr->setPrev(current);

}

void insertToHead(int x)

{

DoublyNode\* ptr = new DoublyNode;

ptr->setData(x);

ptr->setNext(head);

if (head != NULL)

head->setPrev(ptr);

head = ptr;

}

bool isEmpty()

{

if (head == NULL)

return 1;

return 0;

}

int search(int x)

{

current = head;

int count = 0;

while (current != NULL)

{

if (current->getData() == x)

return 1;

current = current->getNext();

count++;

}

return 0;

}

void update(int val1, int val2)

{

current = head;

while (current != NULL)

{

if (current->getData() == val1)

{

current->setData(val2);

return;

}

current = current->next;

}

}

void insertAtIndex(int x, int idx)

{

DoublyNode\* ptr = new DoublyNode;

ptr->setData(x);

if (idx < 0)

{

return;

}

if (idx == 0)

{

ptr->setNext(head);

head = ptr;

}

int count = 0;

current = head;

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

{

count++;

current = current->getNext();

}

if (!current)

{

return;

}

DoublyNode\* ptr2;

ptr2 = current->getPrev();

ptr->setNext(current);

ptr->setPrev(ptr2);

if (ptr2 != nullptr)

{

ptr2->setNext(ptr);

}

current->setPrev(ptr);

}

void deleteData(int x)

{

DoublyNode\* previous = NULL;

current = head;

int count = 0;

while (current != NULL)

{

if (current->getData() == x)

break;

previous = current;

current = current->getNext();

count++;

}

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

{

head = current->getNext();

delete current;

}

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

{

previous->setNext(current->getNext());

delete current;

}

}

void print()

{

current = head;

while (current != NULL)

{

cout << current->data << endl;

current = current->next;

}

}

bool isFull()

{

current = head;

int count = 0;

while (current != NULL)

{

count++;

current = current->next;

}

if (count >= size)

{

cout << "List is full\n";

return 1;

}

cout << "List is not full\n";

return 0;

}

};

int main()

{

DoublyLinkedList list1;

list1.insert(1);

list1.insert(2);

list1.insert(3);

list1.insert(4);

list1.insert(5);

cout << "Values inserted in linked list:\n";

list1.print();

cout << endl;

list1.deleteData(2);

cout << "Deleting value 2\n";

list1.print();

cout << endl;

cout << "is linked list empty?\n";

if (list1.isEmpty())

{

cout << "List is empty\n";

}

else

{

cout << "List is not empty:\n";

list1.print();

}

cout <<endl<< "Inserting 2 more values and then checking if full\n";

list1.insert(6);

list1.insert(2);

list1.isFull();

list1.print();

return 0;

}

Q2:





void removeDuplicate(DoublyLinkedList& list1)

{

list1.current = list1.head;

while (list1.current->next != NULL)

{

if (list1.current->getData() == list1.current->next->getData())

{

list1.current = list1.current->next;

DoublyNode\* ptr = list1.current->next;

list1.deleteData(list1.current->getData());

list1.current = ptr;

}

else

{

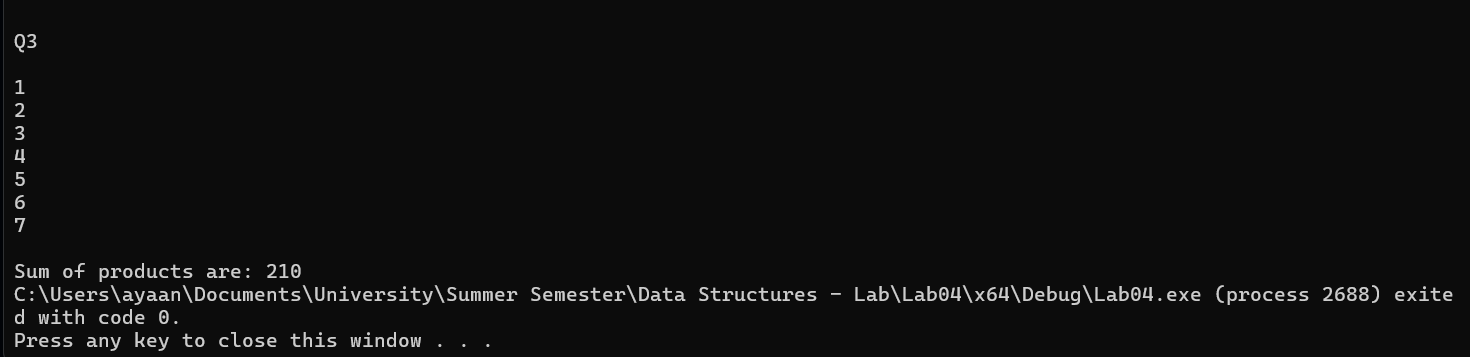
list1.current = list1.current->next;

}

}

}

Q3:







bool isPrime(int x)

{

if (x == 2)

return 1;

int count = 2;

while (count < x)

{

if (x % count == 0)

{

return 0;

}

count++;

}

return 1;

}

int multiplyPrimeNodes(DoublyLinkedList& list1)

{

int count = 1;

list1.current = list1.head;

int sum = 1;

while (list1.current != NULL)

{

if (isPrime(count))

{

sum \*= list1.current->getData();

}

count++;

list1.current = list1.current->next;

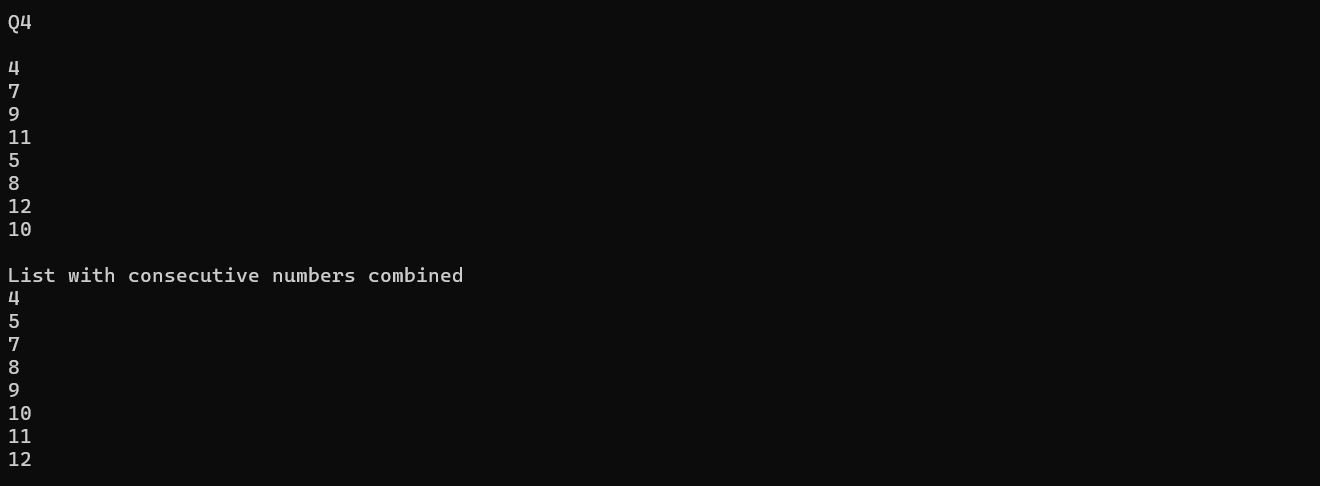
}

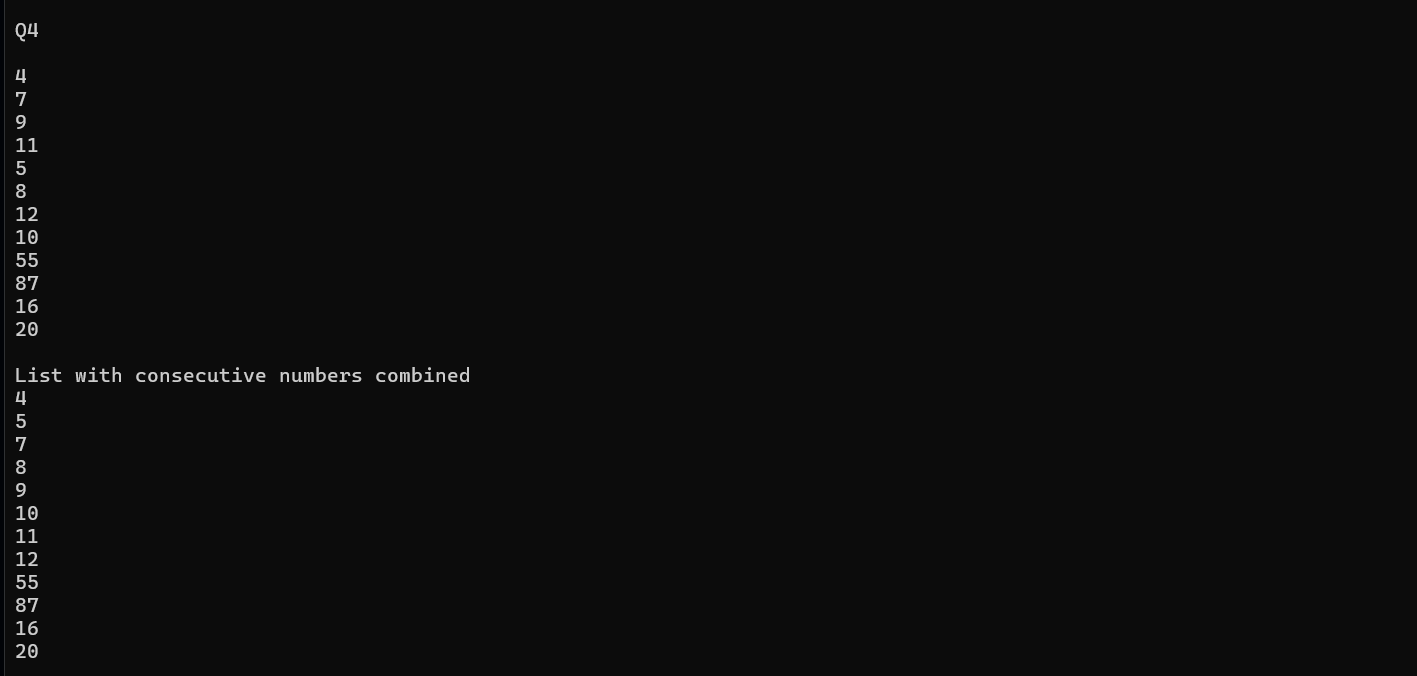
return sum;

}

Q4:







void combineConsecutive(DoublyLinkedList& list1)

{

list1.current = list1.head;

int count = 1;

while (list1.current->next != NULL)

{

DoublyNode\* ptr = list1.current;

if (list1.current->getData() + 1 == list1.current->next->getData())

{

list1.current = list1.current->next;

count++;

continue;

}

else if (list1.search(list1.current->getData() + 1))

{

list1.current = ptr;

list1.deleteData(list1.current->getData() + 1);

list1.current = ptr;

list1.insertAtIndex(list1.current->getData() + 1, count);

list1.current = ptr;

}

list1.current = ptr;

count++;

list1.current = list1.current->next;

}

}

Q5:







DoublyLinkedList mergeLists(DoublyLinkedList& list1, DoublyLinkedList& list2)

{

list1.current = list1.head;

list2.current = list2.head;

while (list1.current->next != NULL)

{

list1.current = list1.current->next;

}

list1.current->next = list2.current;

list2.current->prev = list1.current;

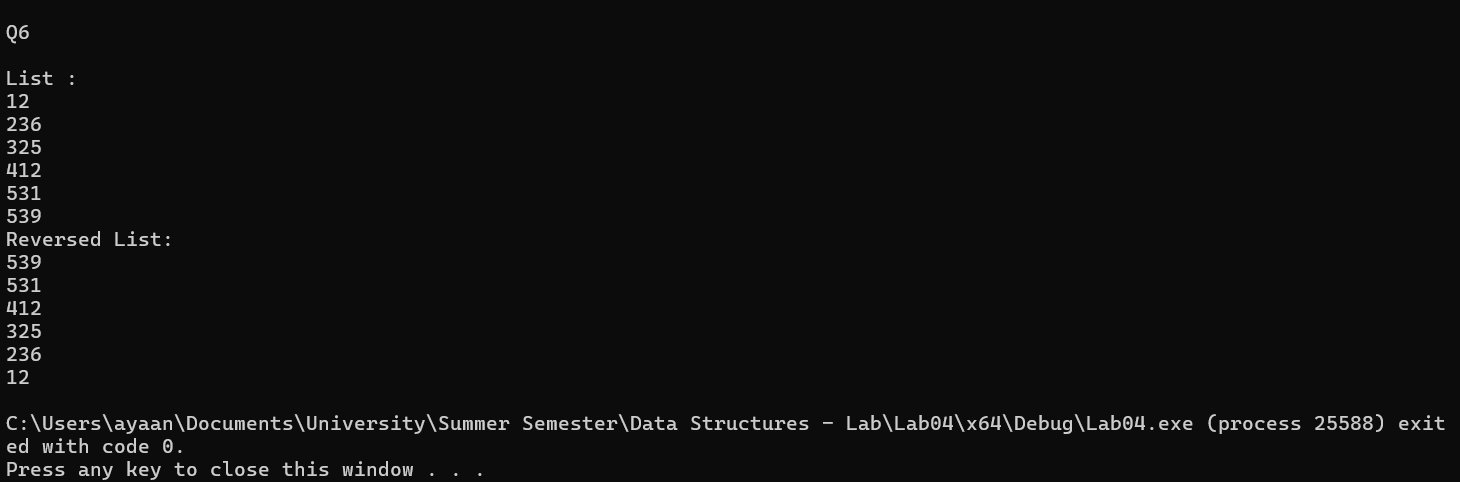
return list1;

}

Q6:







void reverseList(DoublyLinkedList& list1)

{

DoublyLinkedList list2;

list1.current = list1.head;

while (list1.current->next != NULL)

{

list1.current = list1.current->next;

}

list2.head = list1.current;

list2.current = list2.head;

while (list2.current != NULL)

{

list2.current->next = list1.current->prev;

list2.current = list2.current->next;

list1.current = list1.current->prev;

}

list1.head = list2.head;

}