# Applied Programming



# Name: M. Hassan Sattar

# Roll No: 22F-3773

**Section: MS-1A**

# Submitted to: Sir. Muhammad Yousaf

# Problem 1

**Take inputs in a user defined array. After that if the input is even, place it at an even index and if the input is odd, place it an odd index. If the user puts in an even integer and all even indexes are occupied, ask user to enter an odd integer and vice versa. If all the even and odd entries are filled, then notify the user that the program has ended.**

**Inputs: 9 8 5 7 4 2 1**

**Output: 8 9 4 5 2 7 – 1**

**Index: 0 1 2 3 4 5 6 7**

#include <bits/stdc++.h>

using namespace std;

void arrangeEvenAndOddWithIndex(int arr[], int n)

{

    int oddIndex = 1;

    int evenIndex = 0;

    while (true)

    {

        while(evenIndex < n && arr[evenIndex] % 2 == 0)

            evenIndex += 2;

        while(oddIndex < n && arr[oddIndex] % 2 == 1)

            oddIndex += 2;

        if (evenIndex < n && oddIndex < n)

            swap(arr[evenIndex], arr[oddIndex]);

        else

            break;

    }

}

 void displayArray(int arr[], int n)

{

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

}

 int main()

{

    int arr[8] = {};

    cout << "Enter 8 values" << endl;

    for (int i = 0; i < 8; i++)

    {

        cin >> arr[i];

    }

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Input Array :";

    displayArray(arr, n);

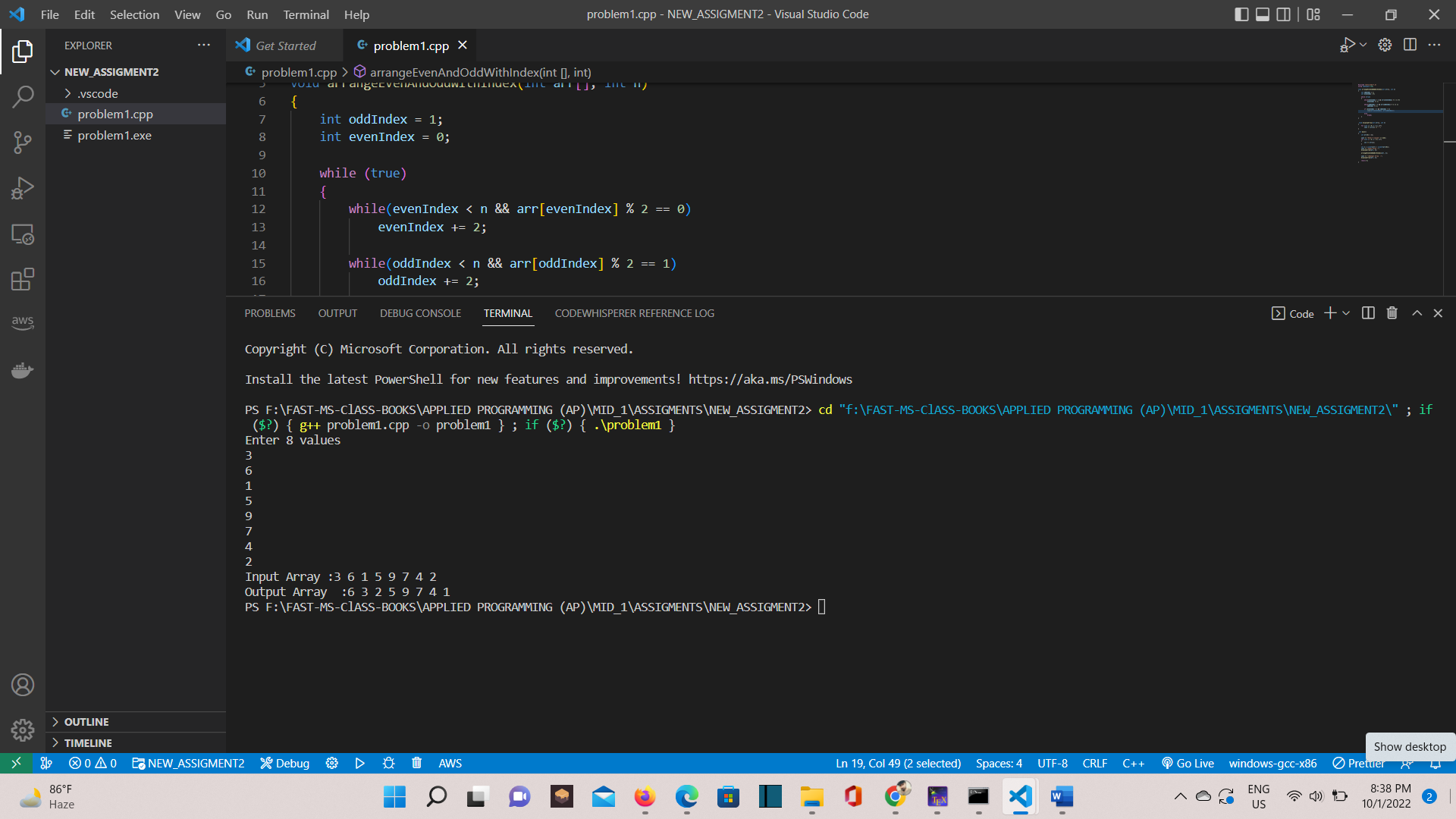
    arrangeEvenAndOddWithIndex(arr, n);

    cout << "\nOutput Array  :";

    displayArray(arr, n);

    return 0;

}

****

# Problem 2

**Create a user defined array and then take inputs. Print all the Divisors of every number user gave as input. Inputs: Size: 3 Elements: 4 5 6 Output: 4: 1 2 5: 1 6: 1 2 3**

#include <bits/stdc++.h>

using namespace std;

void Divisors(int arr[], int n)

{

    int i;

    int j = 0;

    for (j = 0; j < arr[j]; j++)

    {

        cout << "\nAll the divisor of  are: " << arr[j] << "\n";

        for (i = 1; i <= arr[i]; i++)

        {

            if ((arr[j] % i) == 0)

            {

                cout << i << " ";

                cout << "\n";

            }

        }

    }

}

void displayArray(int arr[], int n)

{

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

}

int main()

{

    int arr[3] = {};

    cout << "Enter 3 values" << endl;

    for (int i = 0; i < 3; i++)

    {

        cin >> arr[i];

    }

    int n = sizeof(arr) / sizeof(arr[0]);

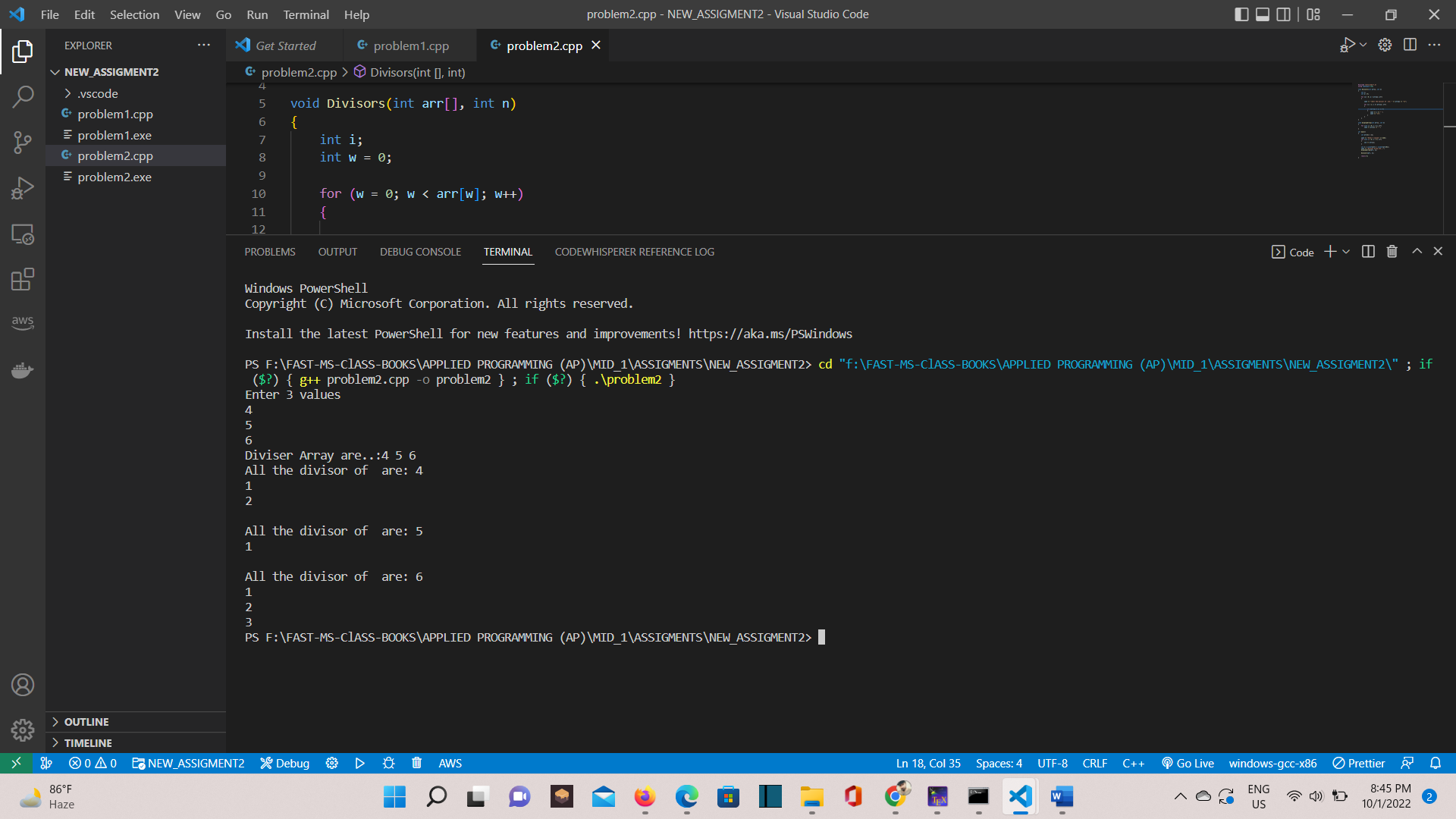
    cout << "Diviser Array are..:";

    displayArray(arr, n);

    Divisors(arr, n);

    return 0;

}



# Problem 3

**Your task is to input two sorted singly linked lists in descending order. Change the next pointers to obtain a single, merged linked list which also has data in descending order. Note that either head pointer given may be null meaning that the corresponding list is empty. You must use the following function prototype, node\* merge\_lists(node \*, node \*);**

#include <iostream>

using namespace std;

struct Node

{

    int data;

    struct Node \*next;

};

Node \*merge\_lists(Node \*a, Node \*b)

{

    if (a == NULL && b == NULL)

        return NULL;

    Node \*res = NULL;

    while (a != NULL && b != NULL)

    {

        if (a->data <= b->data)

        {

            Node \*temp = a->next;

            a->next = res;

            res = a;

            a = temp;

        }

        else

        {

            Node \*temp = b->next;

            b->next = res;

            res = b;

            b = temp;

        }

    }

    while (a != NULL)

    {

        Node \*temp = a->next;

        a->next = res;

        res = a;

        a = temp;

    }

    while (b != NULL)

    {

        Node \*temp = b->next;

        b->next = res;

        res = b;

        b = temp;

    }

    return res;

}

void displayList(struct Node \*Node)

{

    while (Node != NULL)

    {

        cout << Node->data << " ";

        Node = Node->next;

    }

}

Node \*newNode(int data)

{

    Node \*temp = new Node;

    temp->data = data;

    temp->next = NULL;

    return temp;

}

int main()

{

    struct Node \*res = NULL;

    Node \*a = newNode(5);

    a->next = newNode(10);

    a->next->next = newNode(15);

    Node \*b = newNode(2);

    b->next = newNode(3);

    b->next->next = newNode(20);

    cout << "list A before merge: \n";

    displayList(a);

    cout << "\n list B before merge: \n";

    displayList(b);

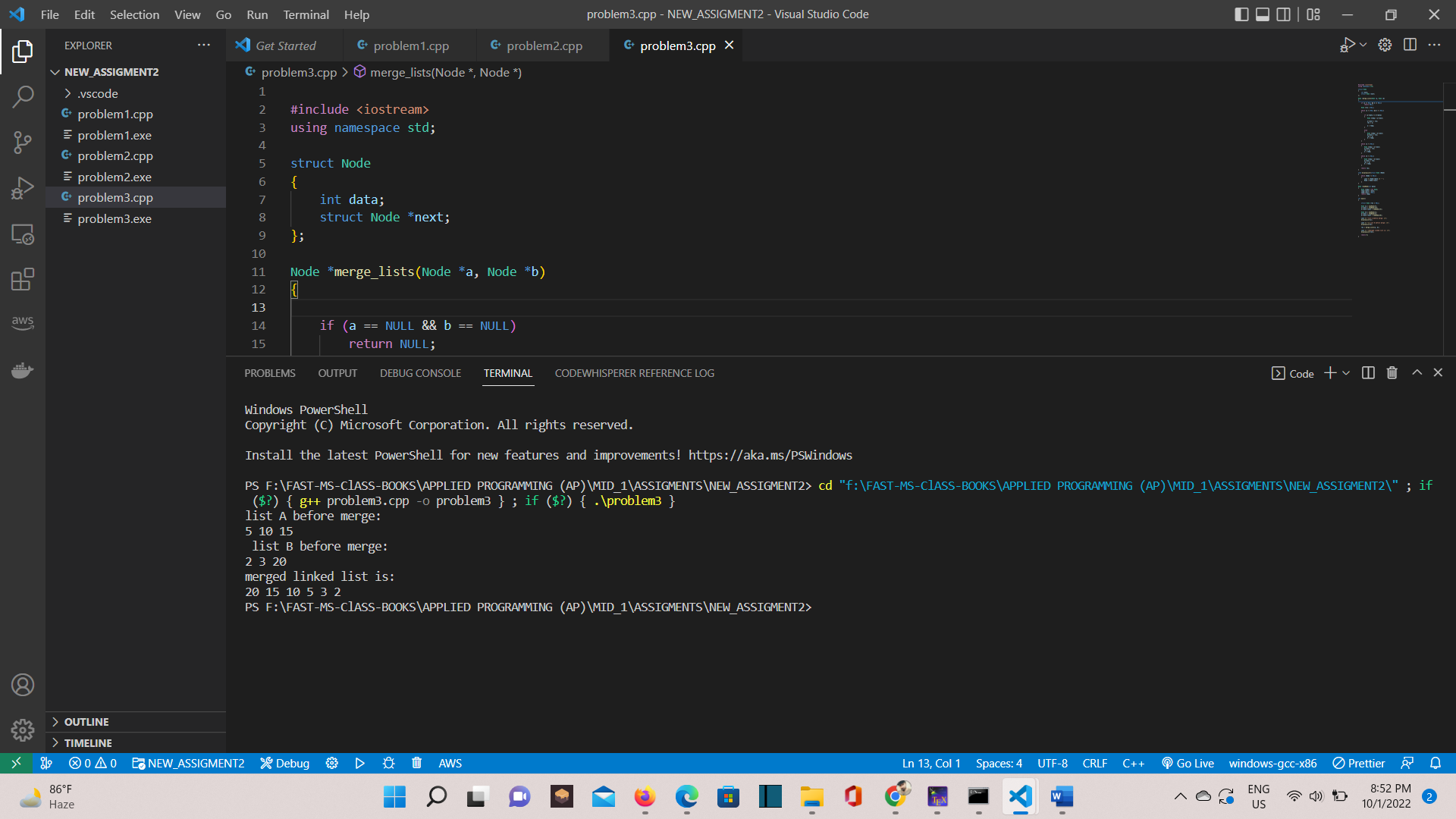
    res = merge\_lists(a, b);

    cout << "\nmerged linked list is: \n";

    displayList(res);

    return 0;

}

****

# Problem 4

**Your task is to take two singly linked lists let suppose A and B. The last index of A is connected to 3rd index of the B. Find the common elements in both linked lists**

#include <iostream>

using namespace std;

struct Node

{

    int data;

    struct Node \*next;

} \*temp = NULL, \*first = NULL, \*second = NULL;

struct Node \*Create(int A[], int n)

{

    int i;

    struct Node \*t, \*last;

    temp = (struct Node \*)malloc(sizeof(struct Node));

    temp->data = A[0];

    temp->next = NULL;

    last = temp;

    for (i = 1; i < n; i++)

    {

        t = (struct Node \*)malloc(sizeof(struct Node));

        t->data = A[i];

        t->next = NULL;

        last->next = t;

        last = t;

    }

    return temp;

}

void Display(struct Node \*p)

{

    while (p != NULL)

    {

        printf("%d ", p->data);

        p = p->next;

    }

}

void Concat(struct Node \*first, struct Node \*second)

{

    struct Node \*p = first;

    while (p->next != NULL)

    {

        p = p->next;

    }

    p->next = second->next->next;

    second->next->next = NULL;

}

void CommonNodes(struct Node \*head1, struct Node \*head2)

{

    struct Node \*current1 = first;

    struct Node \*current2 = second;

    while (current1 != NULL)

    {

        while (current2 != NULL)

        {

            if (current1->data == current2->data)

            {

                cout << "\n";

                cout << "same nodes on both Linked list:  " << current1->data << "  " << endl;

            }

            current2 = current2->next;

        }

        current1 = current1->next;

        current2 = second;

    }

}

int main()

{

    int A[] = {9, 7, 4, 3, 5, 16};

    int B[] = {3, 5, 6, 8, 10, 20};

    first = Create(A, 6);

    second = Create(B, 6);

    cout << "A Linked List: ";

    Display(first);

    cout << "\n B Linked List: ";

    Display(second);

    Concat(first, second);

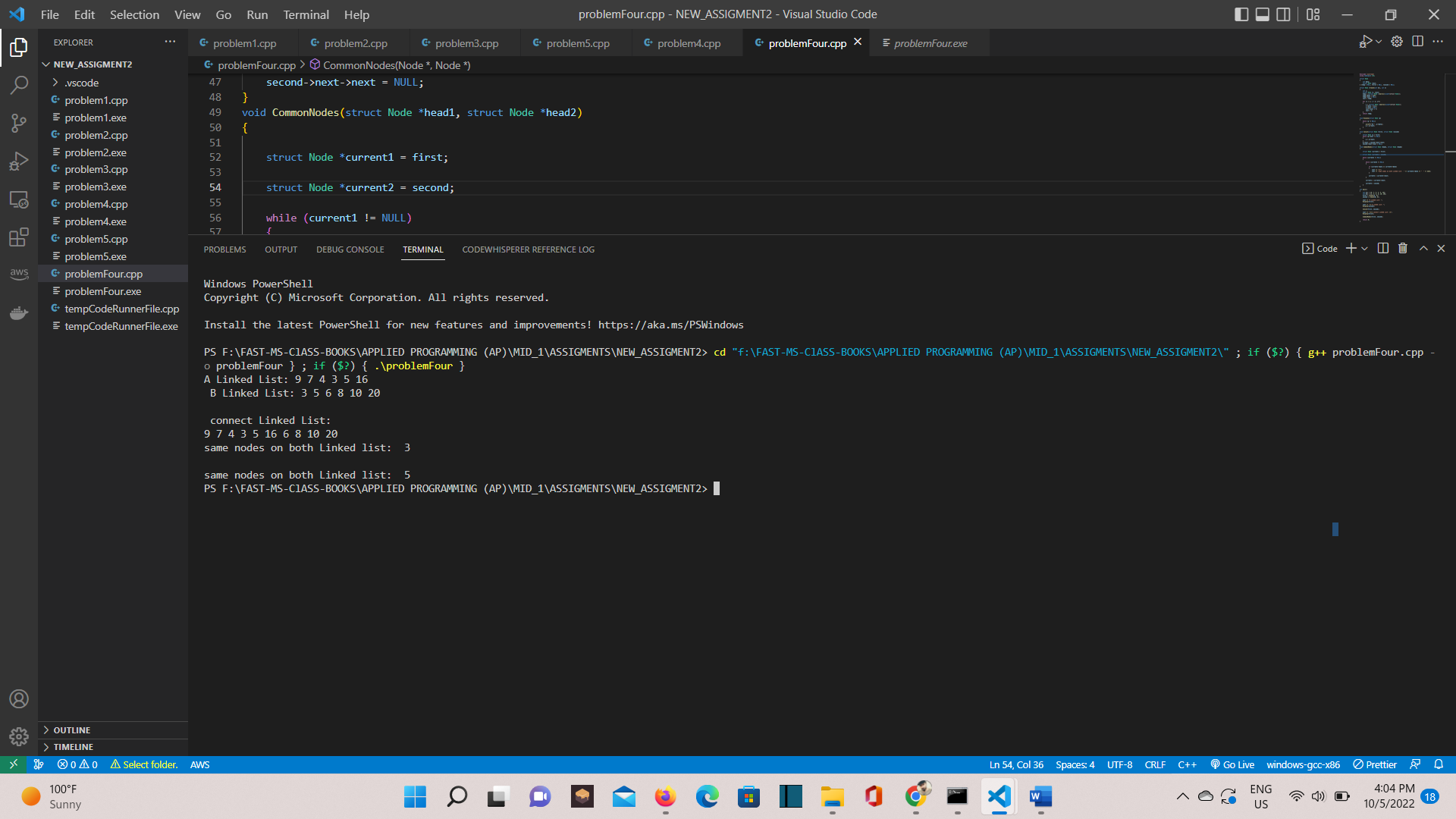
    cout << "\n\n connect Linked List: \n";

    Display(first);

    CommonNodes(first, second);

    return 0;

}



# Problem 5

**Make a doubly linked list and done all the things mention below. The node will have an int variable in the data part. Your LinkedList will have a head and a tail pointer. Your LinkedList class must have the following functions: 1) Insert a node void insertNodeAtBeginning(int data); void insertNodeInMiddle(int key, int data); //will search for key and insert node after the node where a node’s data==key void insertNodeAtEnd(int data); 2) Delete a node bool deleteFirstNode(); //will delete the first node of the LL bool deleteNode(int key); //search for the node where its data==key and delete that particular node bool deleteLastNode(); //will delete the last node of the LL 3) Searching 4) Display**

#include<iostream>

using namespace std;

struct Node

{

    int data;

    Node\* next,\*prev;

};

class doublyList

{

  private:

    Node\*head, \*temp,\*tail;

  public:

    /\*-------------------------------INSERTION---------------------------------------\*/

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

     {

      Node\* node = new Node();

    node->data = value;

    node->next = head;

    if (head != NULL)

        head->prev = node;

    node->prev = NULL;

    head = node;

}

    }

    void insertNodeAtEnd(Node\*& head, int value)

{

    if (head == NULL)

    {

        insertNodeAtBeginning(head, value);

        return;

    }

    Node\* node = new Node();

    node->data = value;

    Node\* temp = head;

    while (temp->next != NULL)

        temp = temp->next;

    temp->next = node;

    node->prev = temp;

    node->next = NULL;

}

   void insertNodeInMiddle(Node\*& head, int value, int after)

{

    Node\* node = new Node();

    node->data = value;

    Node\* temp = head;

    while (temp != NULL && temp->data != after)

        temp = temp->next;

    if (temp == NULL)

    {

        cout << "Key not found. Cannot insert value.\n";

        return;

    }

    node->next = temp->next;

    if(temp->next != NULL)

        temp->next->prev = node;

    temp->next = node;

    node->prev = temp;

}

    /\*-----------------------------DELETION----------------------------\*/

    void deleteFirstNode(Node\*& head){

     Node\* node = head;

    head = head->next;

    head->prev = NULL;

    delete node;

    }

    void deleteLastNode(Node\*& head){

    if (head == NULL)

        return;

    if (head->next == NULL)

    {

        delete head;

        return;

    }

    Node\* temp = head;

    while (temp->next->next != NULL)

        temp = temp->next;

    delete temp->next;

    temp->next = NULL;

    }

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

      Node\* temp = head;

    if (temp->data == key)

    {

        deleteFirstNode(head);

        temp = temp->next;

        return;

    }

    while (temp != NULL && temp->data != key)

        temp = temp->next;

    if (temp == NULL)

    {

        cout << "Node Not found..." << endl;

        return;

    }

    if (temp->next == NULL)

    {

        deleteLastNode(head);

    }

    else

    {

        temp->next->prev = temp->prev;

        temp->prev->next = temp->next;

    }

    }

 /\*--------------------------SEARCHING------------------------------\*/

bool search(Node \*head, int key)

{

  Node \*temp = head;

  int pos = 0;

  if (temp != NULL)

  {

    while (temp != NULL)

    {

      pos++;

      if (temp->data == key)

      {

        cout << temp->data << endl;

        break;

      }

      temp = temp->next;

    }

  }

}

     /\*--------------------------REVERSE------------------------------\*/

Node \*reverve(Node \*&head)

{

  Node \*prevptr = NULL;

  Node \*currptr = head;

  Node \*nextptr;

  while (currptr != NULL)

  {

    nextptr = currptr->next;

    currptr->next = prevptr;

    prevptr = currptr;

    currptr = nextptr;

    /\* code \*/

  }

  return prevptr;

}

    /\*--------------------------DISPLAY------------------------------\*/

    void display(Node\* head)

{

    Node\* temp = head;

    cout << "[ ";

    while (temp != NULL)

    {

        cout << temp->data << " ->";

        temp = temp->next;

    }

    cout << "]\n";

}

};

int main()

{

     Node\* head = NULL;

    doublyList obj;

          /\*----------------------------------------INSERTION--------------------------------\*/

    obj.insertNodeAtBeginning(head,1);

    obj.display(head);

      obj.insertNodeAtBeginning(head,2);

    obj.display(head);

    obj.insertNodeAtEnd(head,3);

     obj.display(head);

    obj.insertNodeAtEnd(head,4);

     obj.display(head);

      obj.insertNodeAtEnd(head,5);

     obj.display(head);

      obj.insertNodeAtEnd(head,6);

     obj.display(head);

    obj.insertNodeInMiddle(head,8,1);

    obj.display(head);

   /\*----------------------------------------DELETION--------------------------------\*/

    obj.deleteFirstNode(head);

    obj.display(head);

    obj.deleteLastNode(head);

    obj.display(head);

    obj.deleteNode(head,8);

    obj.display(head);

          /\*--------------------------SEARCHIING--------------------------------------\*/

    obj.search(head, 1);

    obj.display(head);

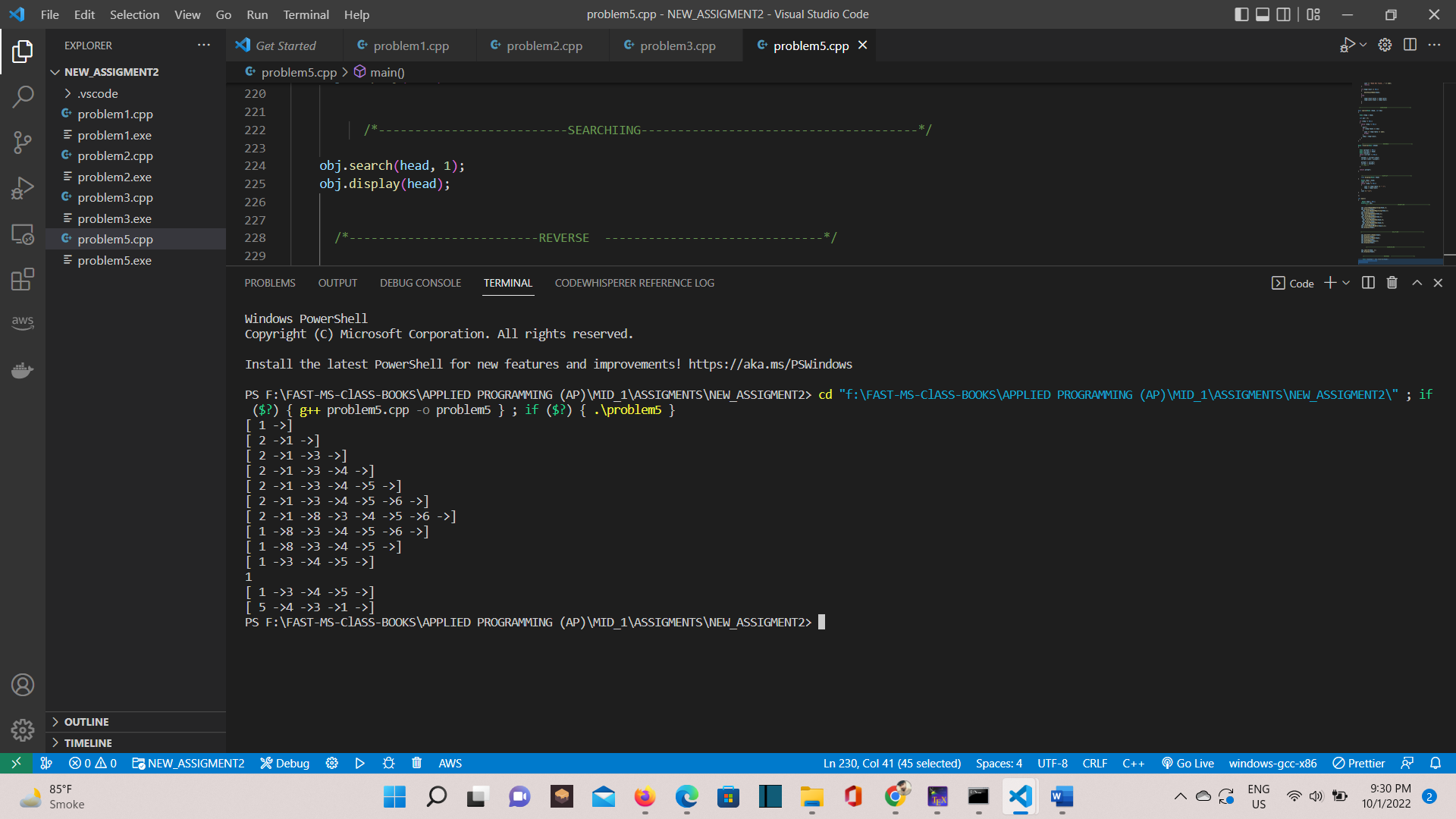
      /\*--------------------------REVERSE  ------------------------------\*/

      Node \*newhead = obj.reverve(head);

    obj.display(newhead);

    return 0;

}

****

# Problem 6

**Write a program that prompts the user to input a string and then outputs the string in the pigLatin form. Input the string in a doubly circular link-list. The rules for converting a string into pig Latin form are as follows: If the string begins with a vowel, add the string "-way" at the end of the string. Forexample, the pig Latin form of the string "eye" is "eye-way". If the string does not begin with a vowel, first add "-" at the end of the string. Then rotate the string one character at a time; that is, move the first character of the string tothe end of the string until the first character of the string becomes a vowel. Then add the string "ay" at the end. For example, the pig Latin form of the string "There" is "ere-Thay Strings such as "by" contain no vowels. In cases like this, the letter y can be considered a vowel. So, for this program the vowels are a, e, i, o, u, y, A, E, I, O, U,and Y. Therefore, the pig Latin form of "by" is "y-bay". Strings such as "1234" contain no vowels. The pig Latin form of the string "1234" is"1234- way". That is, the pig Latin form of a string that has no vowels in it is the string followed by the string "-way"**

#include <iostream>

using namespace std;

struct Node

{

    char data;

    Node\* next, \* prev;

};

void Insert(Node\*& head, char value)

{

    Node\* node = new Node();

    node->data = value;

    if (head == NULL)

    {

        node->next = node->prev = node;

        head = node;

    }

    else

    {

        Node\* last = head->prev;

        node->next = head;

        head->prev = node;

        node->prev = last;

        last->next = node;

    }

}

void PrintList(Node\*& head)

{

    Node\* ptr = head;

    if (ptr != NULL)

    {

        do

        {

            cout << ptr->data;

            ptr = ptr->next;

        } while (ptr != head);

    }

}

void PigLatin(Node\*& head, string str)

{

    string vowel = "aeiouyAEIOUY";

    for (int i = 0; i < str.length(); i++)

    {

        Insert(head, str[i]);

    }

    for (int i = 0; i < vowel.length(); i++)

    {

        if (vowel[i] == head->data)

        {

            Insert(head, '-');

            Insert(head, 'w');

            Insert(head, 'a');

            Insert(head, 'y');

            return;

        }

    }

    bool check\_for\_vowel = false;

    Node\* ptr = head;

    do

    {

        for (int i = 0; i < vowel.length(); i++)

        {

            if (vowel[i] == ptr->data)

            {

                check\_for\_vowel = true;

                break;

            }

        }

        if (check\_for\_vowel)

            break;

        ptr = ptr->next;

    } while (ptr != head);

    if (!check\_for\_vowel)

    {

        Insert(head, '-');

        Insert(head, 'w');

        Insert(head, 'a');

        Insert(head, 'y');

        return;

    }

    Insert(head, '-');

    ptr = head;

    do

    {

        for (int i = 0; i < vowel.length(); i++)

        {

            if (vowel[i] == ptr->data)

            {

                head = ptr;

                Insert(head, 'a');

                Insert(head, 'y');

                return;

            }

        }

        ptr = ptr->next;

    } while (ptr != head);

}

int main()

{

    Node\* node = NULL;

    cout << "Running Test case for: eye\nResult: ";

    PigLatin(node, "eye");

    PrintList(node);

    cout << endl;

    Node\* node2 = NULL;

    cout << "Running Test case for: There\nResult: ";

    PigLatin(node2, "There");

    PrintList(node2);

    cout << endl;

    Node\* node3 = NULL;

    cout << "Running Test case for: 1234\nResult: ";

    PigLatin(node3, "1234");

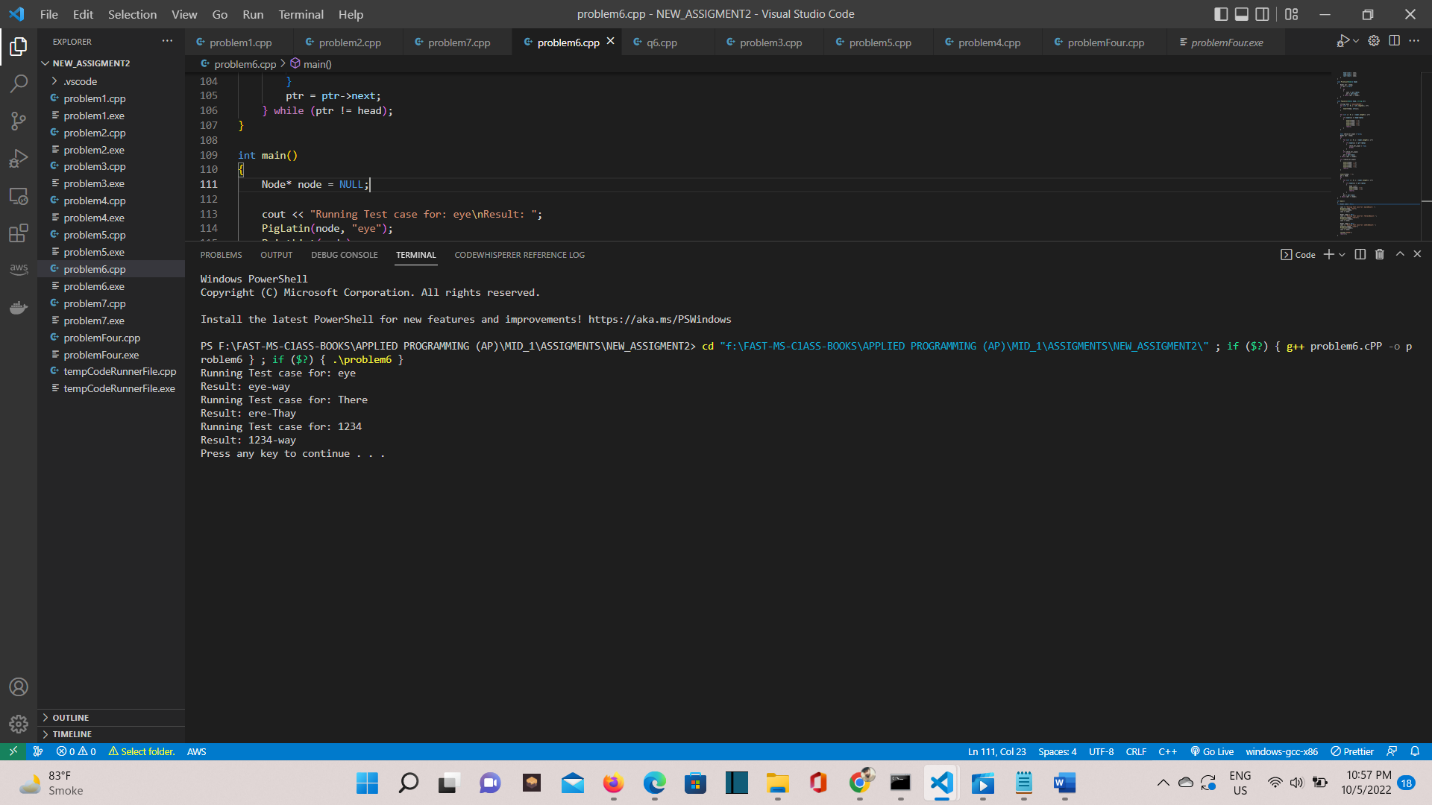
    PrintList(node3);

    cout << endl;

    system("PAUSE");

    return 0;

}

****

# Problem 7

**Round-Robin algorithm is an algorithm used in CPU Scheduling. In this algorithm, all processes take turns and run for a specific interval, until they are all completely executed.You are required to make a process that will consist of two things, ID (must be unique) Execution time Each process will take a certain amount of time to execute completely, this time will bestored in the “execution time” variable. You are required to implement a circular linked list that will store user-entered number of processes (say n). You will also input a time slice from the user, this will be the time every process will get in one turn. You will then create a function that will execute every process in the circular linked list using Round-Robin algorithm and store the total time taken by each process to execute. Your final output will be the time required by each process to completely execute**

#include <iostream>

using namespace std;

struct Node

{

    int ID, time;

    Node\* next;

};

class ProcessControl

{

private:

    Node\* head;

public:

    ProcessControl();

    void Insert(int ID, int time);

    void Delete(int ID);

    void DisplayTotalProcess();

    void Process(int Timeslice);

};

ProcessControl::ProcessControl()

{

    this->head = NULL;

}

void ProcessControl::Insert(int ID, int time)

{

    Node\* node = new Node;

    node->ID = ID;

    node->time = time;

    if (head == NULL)

    {

        head = node;

        head->next = head;

    }

    else

    {

        Node\* ptr = head;

        while (ptr->next != head)

        {

            ptr = ptr->next;

        }

        ptr->next = node;

        node->next = head;

    }

}

void ProcessControl::Delete(int ID)

{

    if (head == NULL)

        return;

    if (head->ID == ID && head->next == head)

    {

        delete head;

        head = NULL;

        return;

    }

    Node\* last = head, \*delete\_node;

    if (head->ID == ID)

    {

        while (last->next != head)

            last = last->next;

        last->next = head->next;

        delete head;

        head = last->next;

        return;

    }

    while (last->next != head && last->next->ID != ID)

        last = last->next;

    if (last->next->ID == ID)

    {

        delete\_node = last->next;

        last->next = delete\_node->next;

        delete delete\_node;

    }

    else

    {

        cout << "No Node Found...\n";

    }

}

void ProcessControl::DisplayTotalProcess()

{

    Node\* ptr = head;

    cout << "Processes in Linked List: ";

    do

    {

        cout << ptr->ID << " ";

        ptr = ptr->next;

    } while (ptr != head);

}

void ProcessControl::Process(int Timeslice)

{

    int timetaken = 0;

    Node\* ptr = head;

    bool flag = false;

    while (head != NULL)

    {

        flag = false;

        do

        {

            DisplayTotalProcess();

            if (ptr->time <= Timeslice && (ptr->time - Timeslice <= 0))

            {

                cout << "Process #" << ptr->ID << " executed for " << ptr->time << " seconds" << endl;

                timetaken += ptr->time;

                cout << "Process #" << ptr->ID << " has been executed. Time Taken: " << timetaken  << endl;

                Delete(ptr->ID);    //   Delete Current Node...

                ptr = head;

                continue;

            }

            else

            {

                cout << "Process #" << ptr->ID << " executed for " << Timeslice << " seconds" << endl;

                ptr->time -= Timeslice;

                timetaken += Timeslice;

            }

            ptr = ptr->next;

        } while (ptr != head);

    }

    cout << "Execution Completed" << endl;

}

int main()

{

    ProcessControl d\_obj;

    int processes, time;

    cout << "Enter number of processes: ";

    cin >> processes;

    for (int i = 1; i <= processes; i++)

    {

        cout << "Enter Process #" << i << " Execution Time: ";

        cin >> time;

        d\_obj.Insert(i, time);

    }

    int timeslice;

    cout << "Enter Time Slice: ";

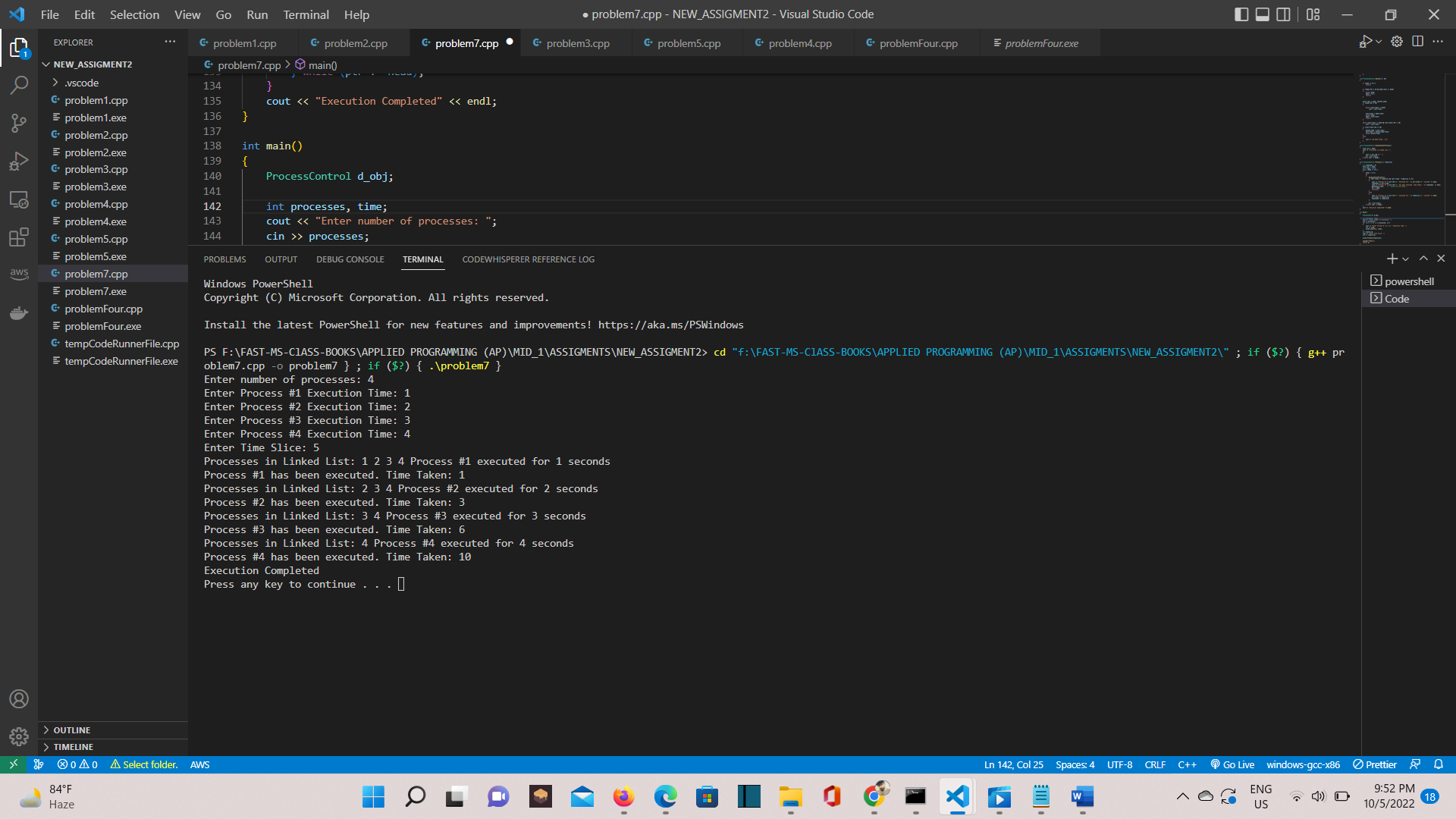
    cin >> timeslice;

    d\_obj.Process(timeslice);

    system("PAUSE");

    return 0;

}

****