**Họ và Tên: Trần Thị Ngọc Diệp**

**MSSV: 1827005**

**Lớp: B2HK182 – Cấu Trúc Dữ Liệu và Giải Thuật**

**(TUT 2 & LAB 2)**

------------------------

**TUT 2**

**Question 1:**

**a. Print the first prime number (if there are prime numbers):**

void onePrime(int \*arr, int n) {

static bool flag; //False: there is no prime number in array

if (n==0){

if (flag!=true){

cout<<"There's no prime number in the array. ";

}

}

else if (isPrime(\*arr)==true){

flag = true;

cout<<"The prime number is: "<<\*arr<<"\n";

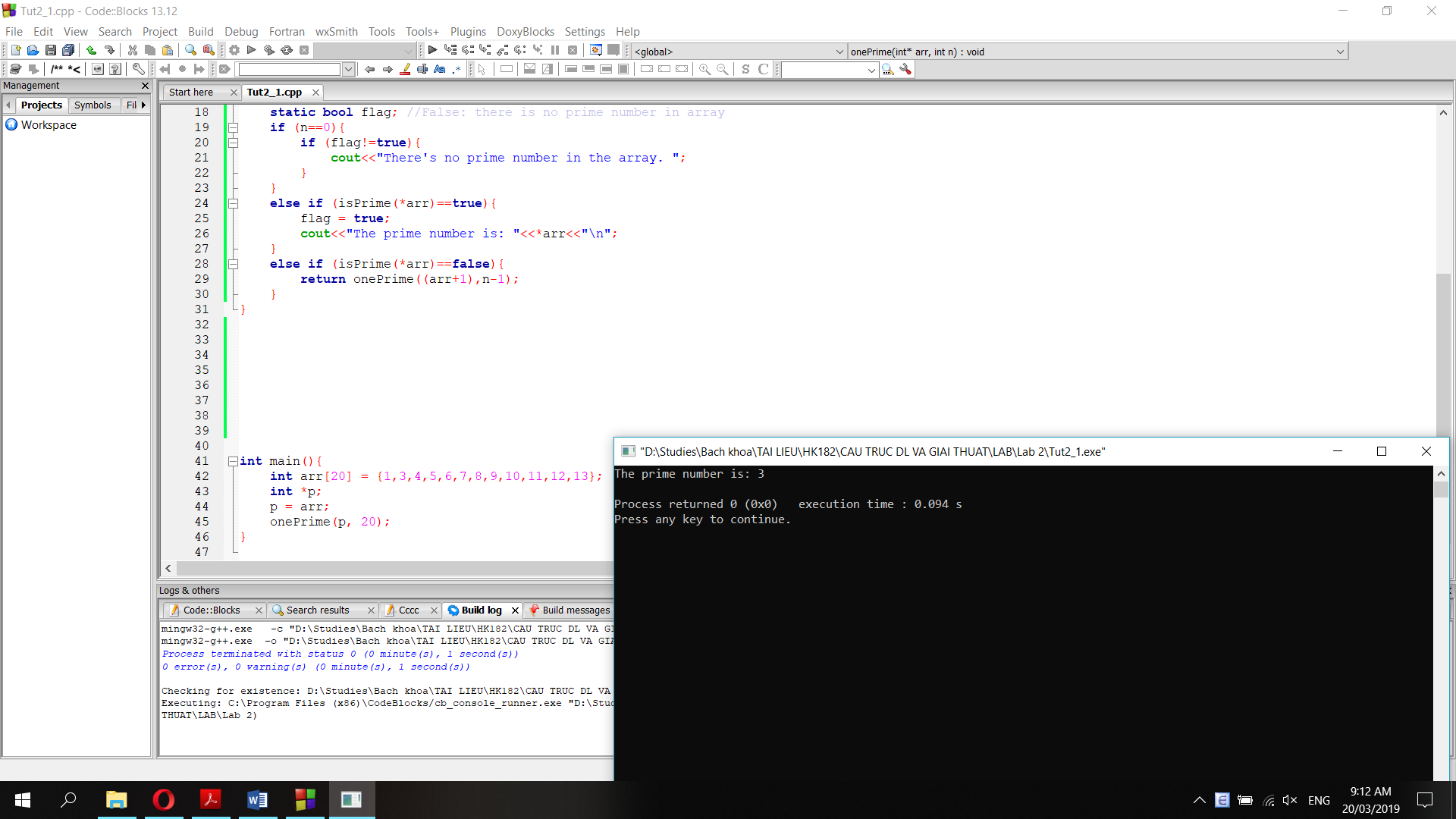
}

else if (isPrime(\*arr)==false){

return onePrime((arr+1),n-1);

}

}



**b.**

void onePrime(int \*arr, int n) {

static bool flag; //False: there is no prime number in array

if (n==0){

if (flag!=true){

cout<<"There's no prime number in the array. ";

}

}

else if (isPrime(\*arr)==true){

flag = true;

cout<<"The prime number is: "<<\*arr<<"\n";

return onePrime((arr+1),n-1);

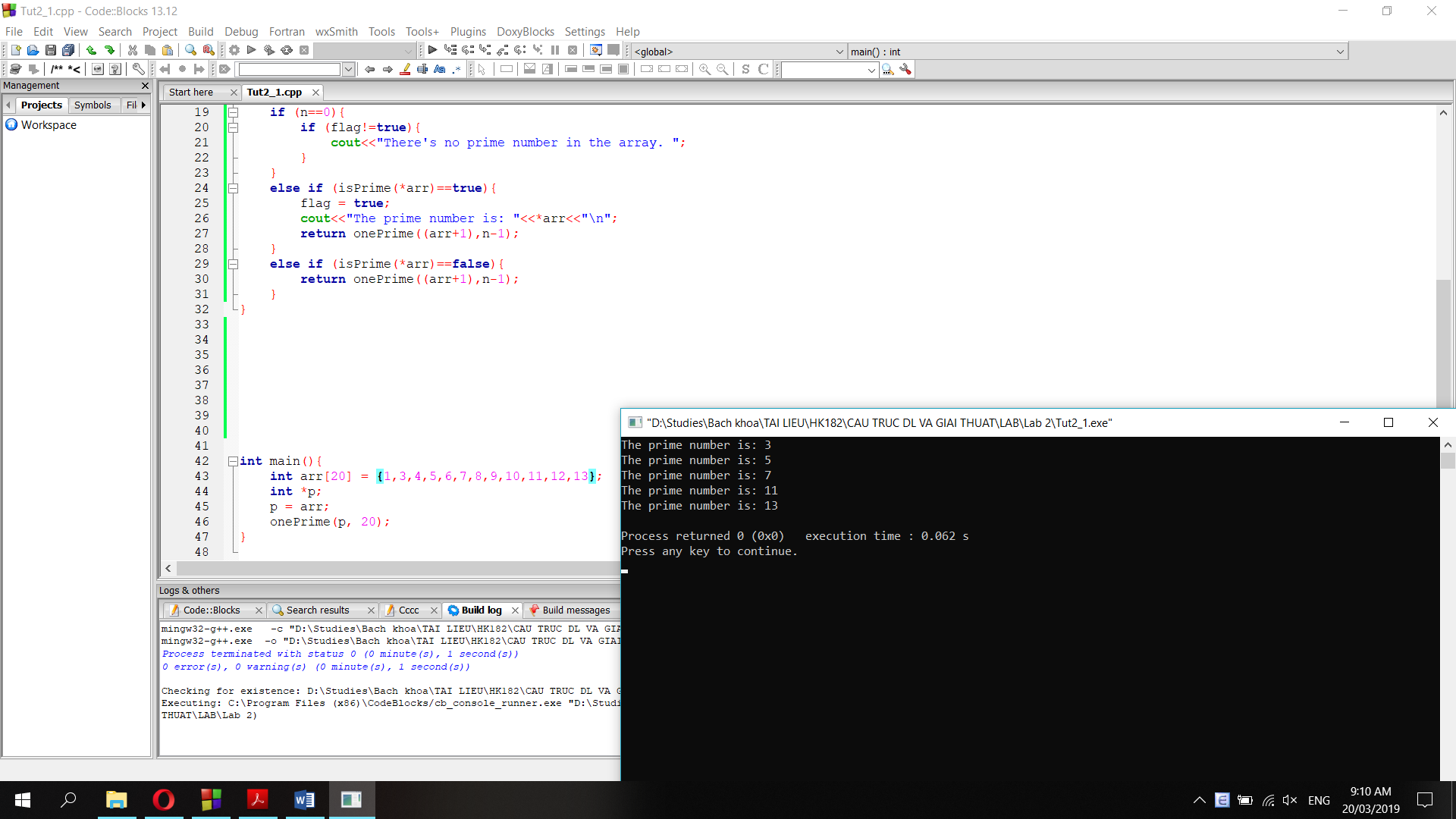
}

else if (isPrime(\*arr)==false){

return onePrime((arr+1),n-1);

}

}



**Question 2:**

int myMaxFunc(Node\* head, int maxVal) {

if (head->next==NULL){

if (head->data>maxVal){

return head->data;

}

else return maxVal;

}

else {

if (head->data>maxVal){

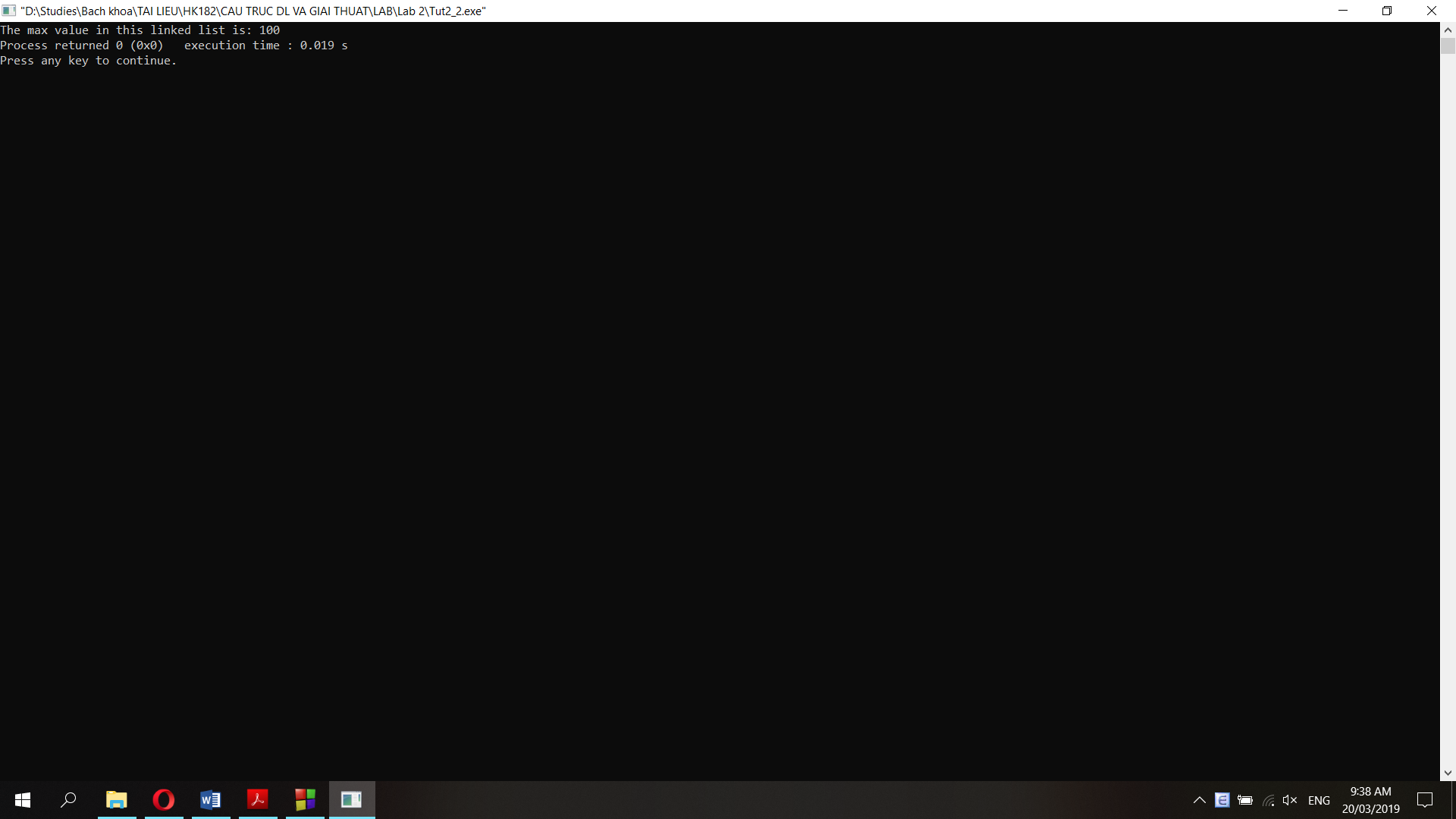
maxVal = head->data;

}

return myMaxFunc(head->next, maxVal);

}

}



**Question 3:**

**a.**

**b.**

void LoopTriangle(int N){

static int n = N;

if (N==0){

return;

}

if (N<n/2+1){

for (int i=1; i<n-N+1; i++){

cout<<"\_";

}

for (int j=n-N+1; j<=n; j++){

cout<<j;

}

cout<<"\n";

return LoopTriangle(N-1);

}

else {

for (int i=1; i<N; i++){

cout<<"\_";

}

for (int j=N; j<=n; j++){

cout<<j;

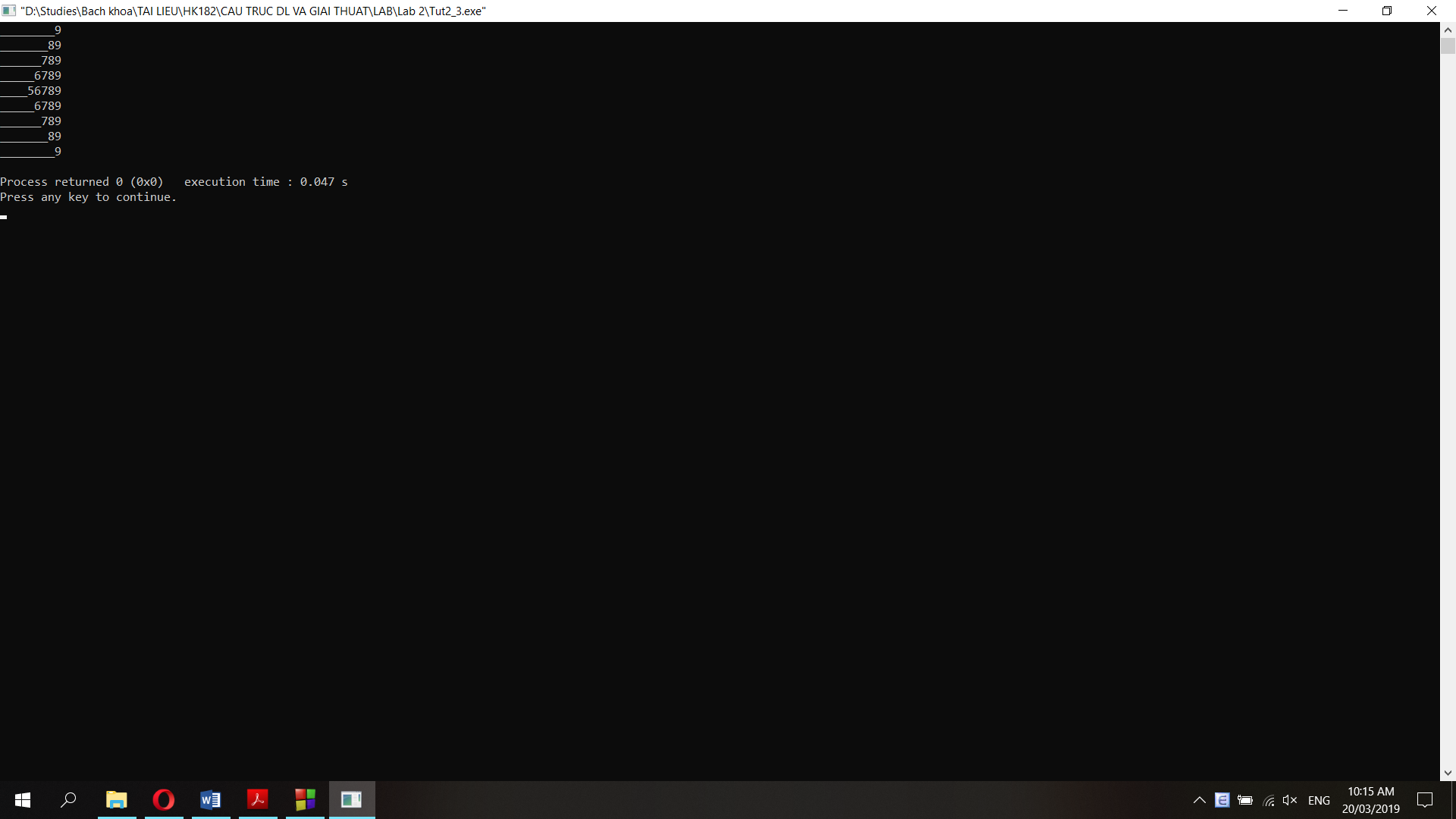
}

cout<<"\n";

return LoopTriangle(N-1);

}

}



**Question 4:**

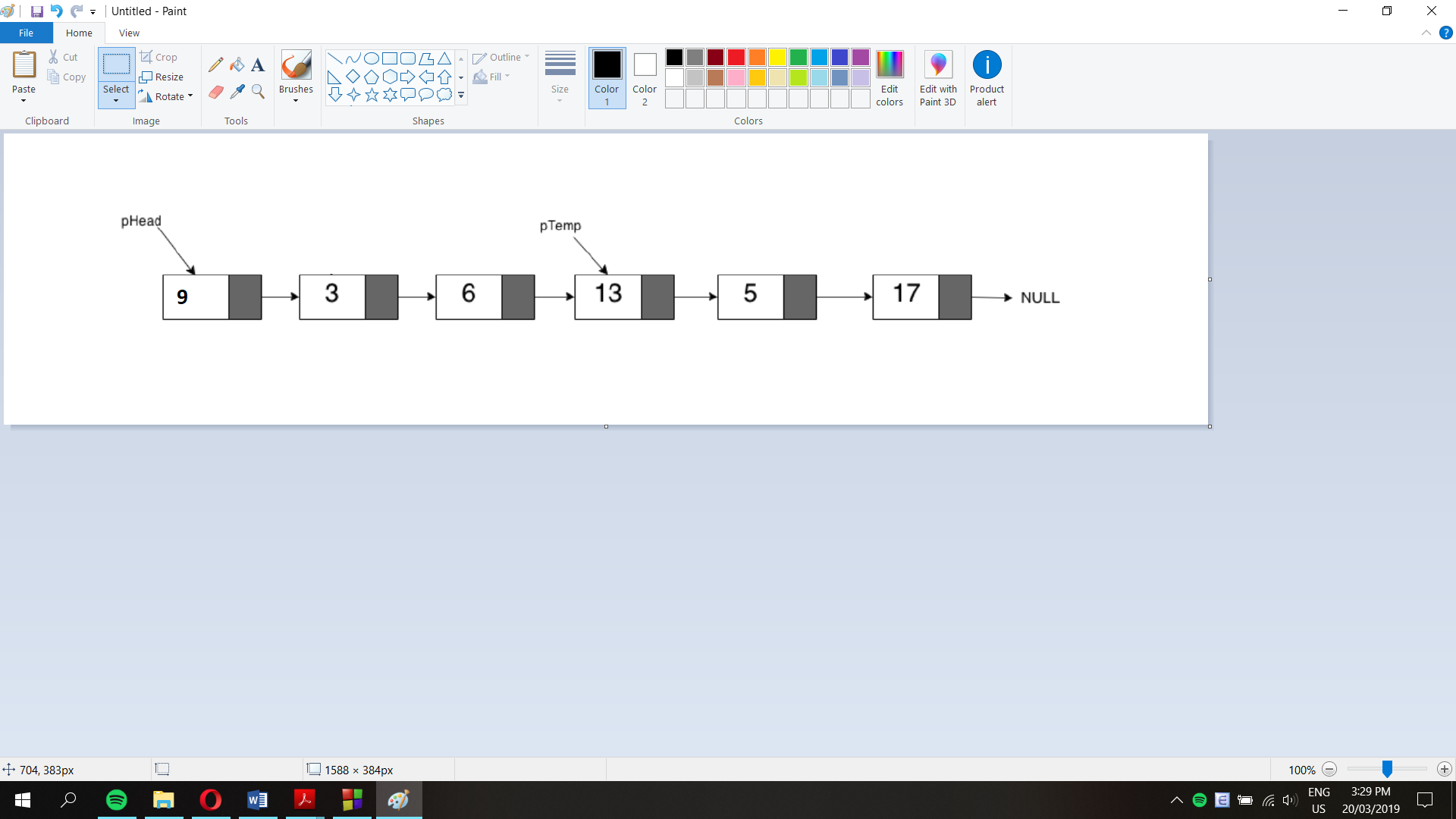
**a. Insert 1 node (data = 9) at the beginning:**

pNew = new Node;

pNew->data = 9;

pNew->next = pHead;

pHead = pNew;



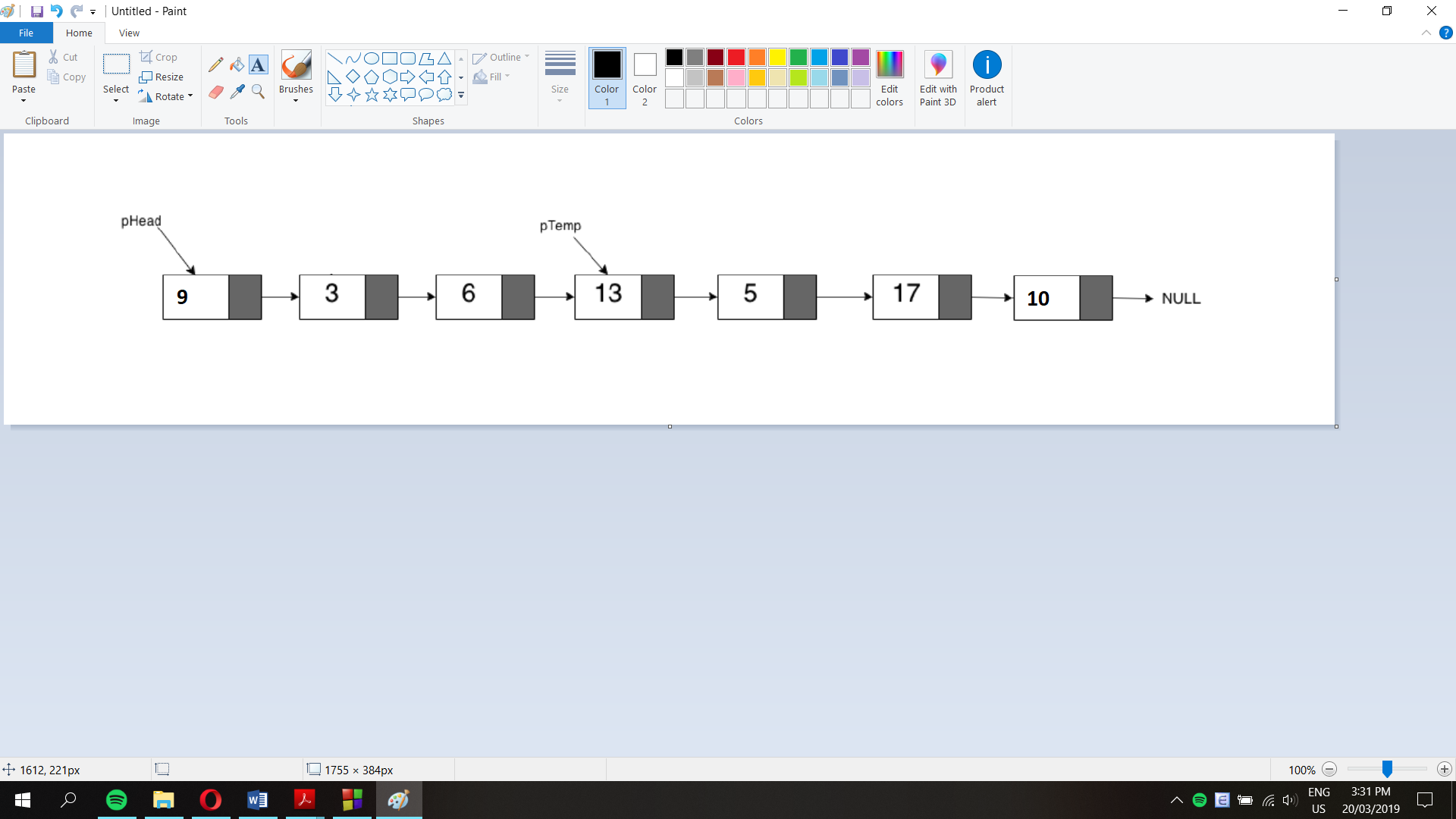
**b. Insert 1 node (data = 10) at the end:**

pNew = new Node;

pNew->data = 10;

pTail->next = pNew;

pTail = pNew;



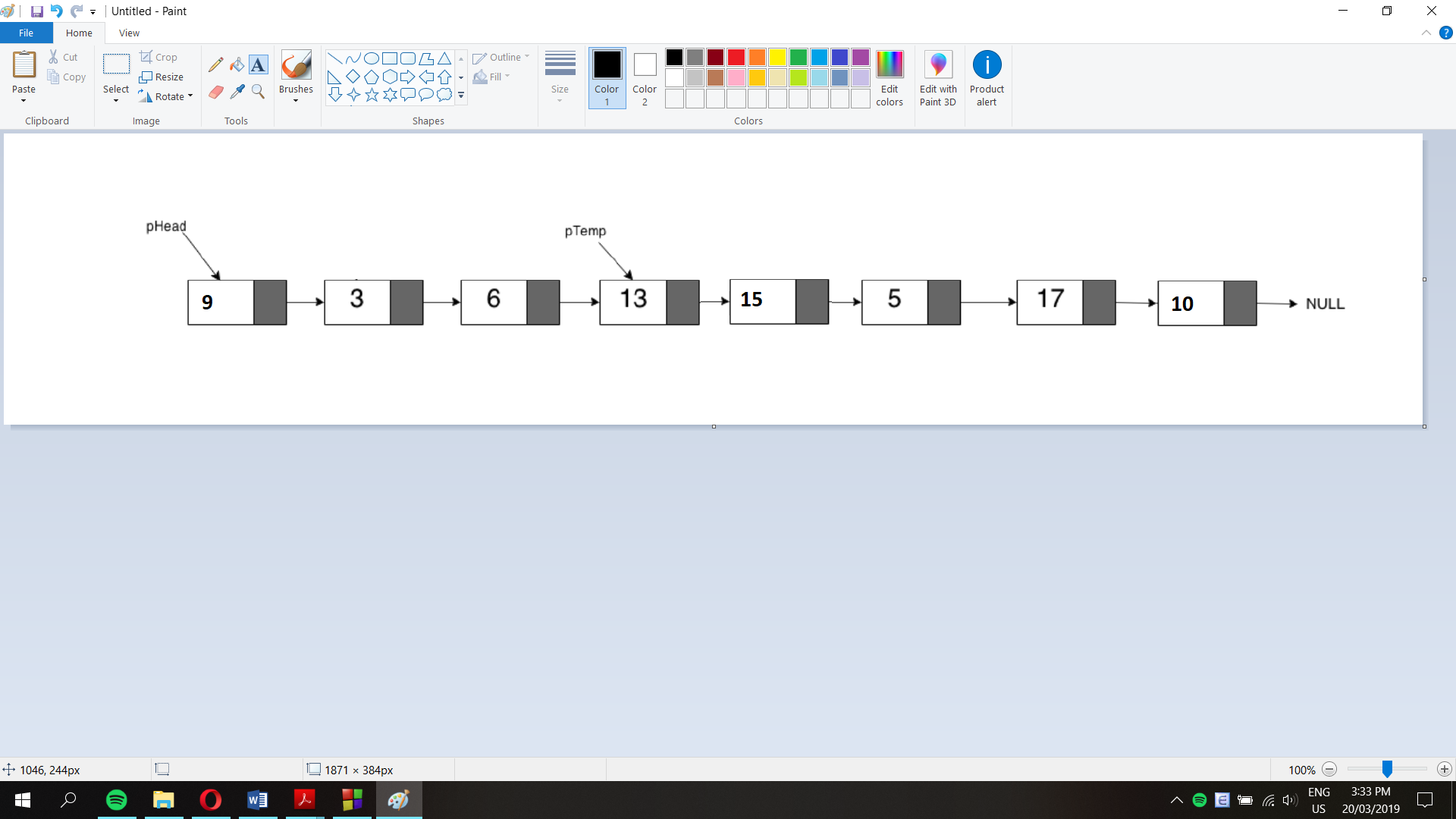
**c. Insert 1 node (data = 15) after pTemp:**

pNew = new Node;

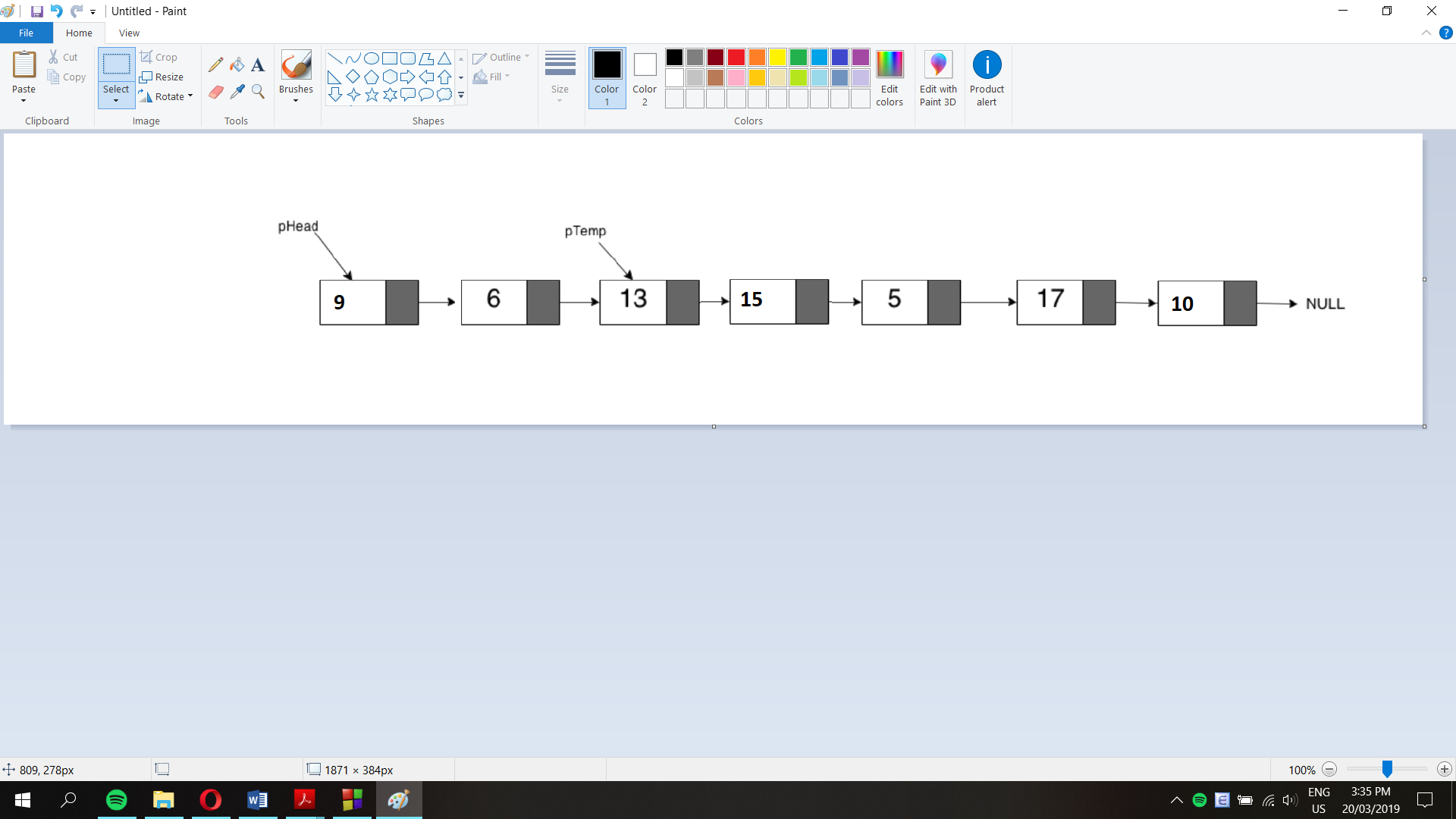
pNew->data = 15;

pNew->next = pTemp->next;

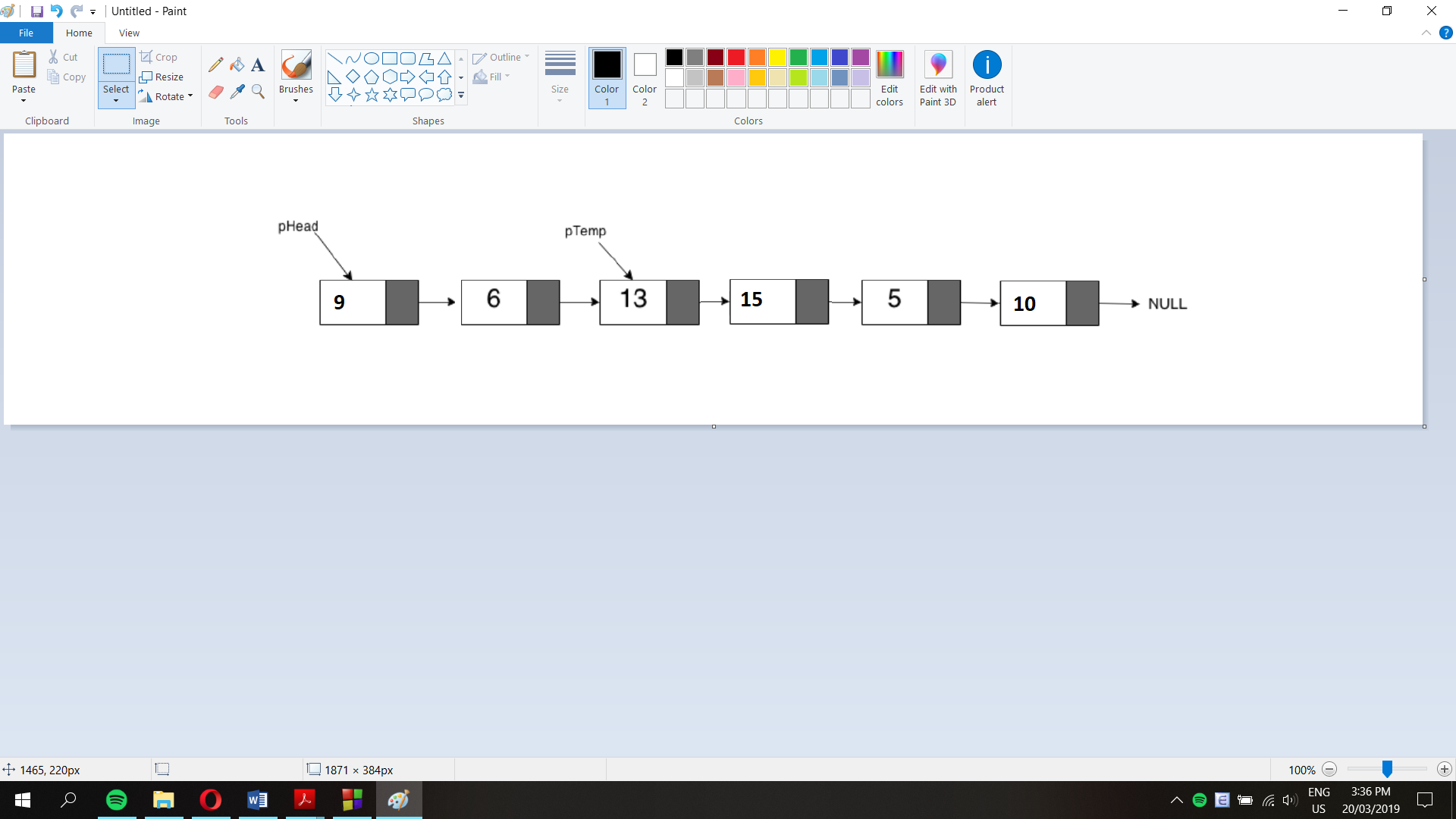
pTemp->next = pNew;



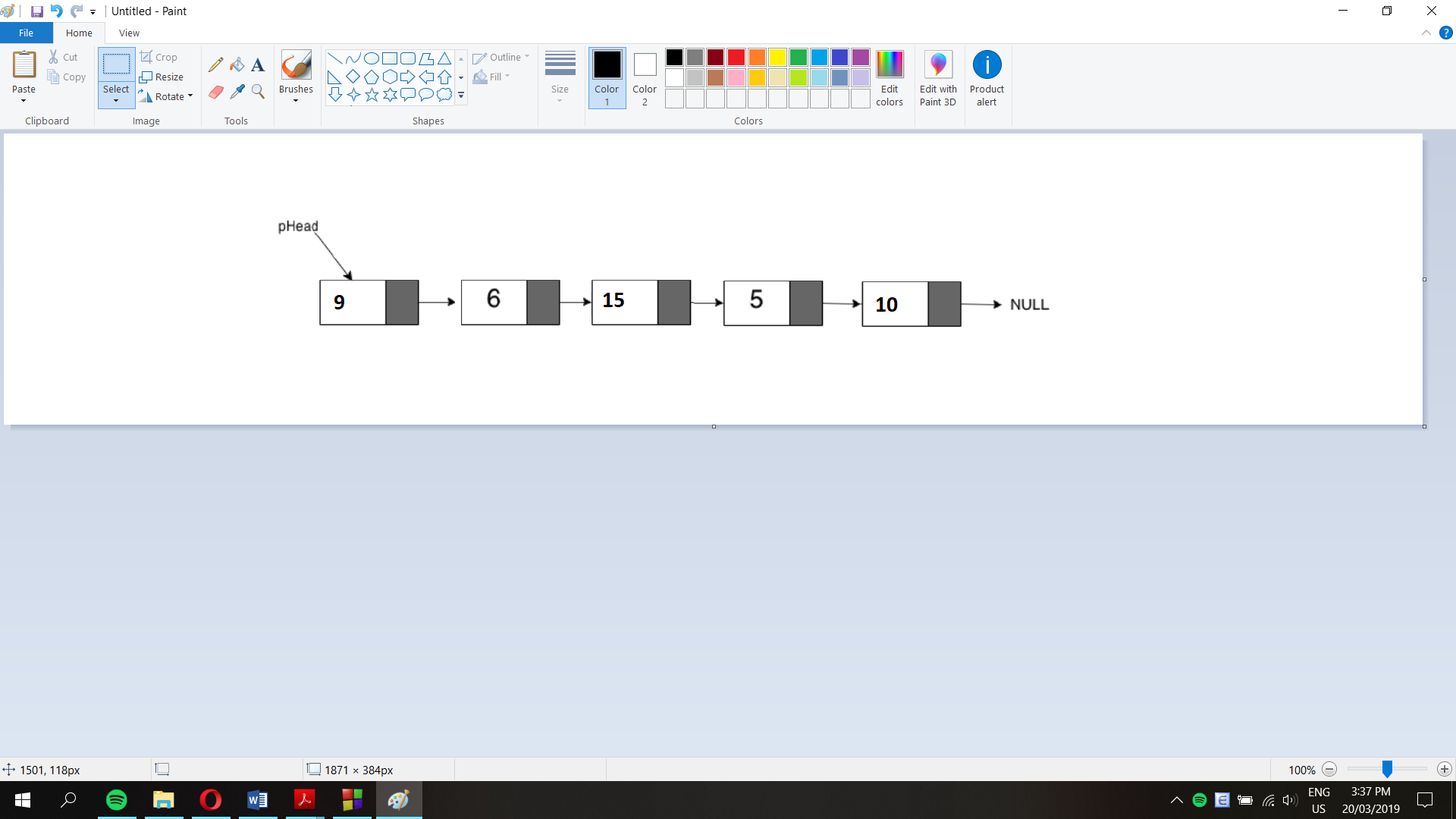
**d. Delete the node which has the data = 3:**



**e. Delete the node which has the data = 17:**



**f. Delete the node which pTemp points to:**



**g.**

void fun1(node\* head)

{

if (head == NULL)

return;

fun1(head->next);

printf("%d ", head->data);

}

Hàm fun1() in các giá trị của linked list theo thứ tự từ pTail đến pHead (bởi vì câu lệnh printf được đặt sau lời gọi hàm hồi quy fun1(head->next).

**h.**

void fun2(node\* head)

{

if (head == NULL)

return;

printf("%d ", head->data);

if (head->next != NULL)

fun2(head->next->next);

printf("%d ", head->data);

}

Hàm fun2() in giá trị các phần tử ở vị trí chẵn (0, 2, 4, 6) của linked list, tuy nhiên bởi vì có thêm câu lệnh printf(“%d “, head->data) ở cuối nên các giá trị ở vị trí chẵn được in ra lại 1 lần nữa qua mỗi lần gọi hàm hồi quy theo thứ tự ngược lại (6,4,2,0).

**Question 5:**

**a. Function prints out all values of a linked list**

void printNode(Node \*Head){

if (Head->next == NULL){

cout<<Head->data<<" ";

return;

}

else {

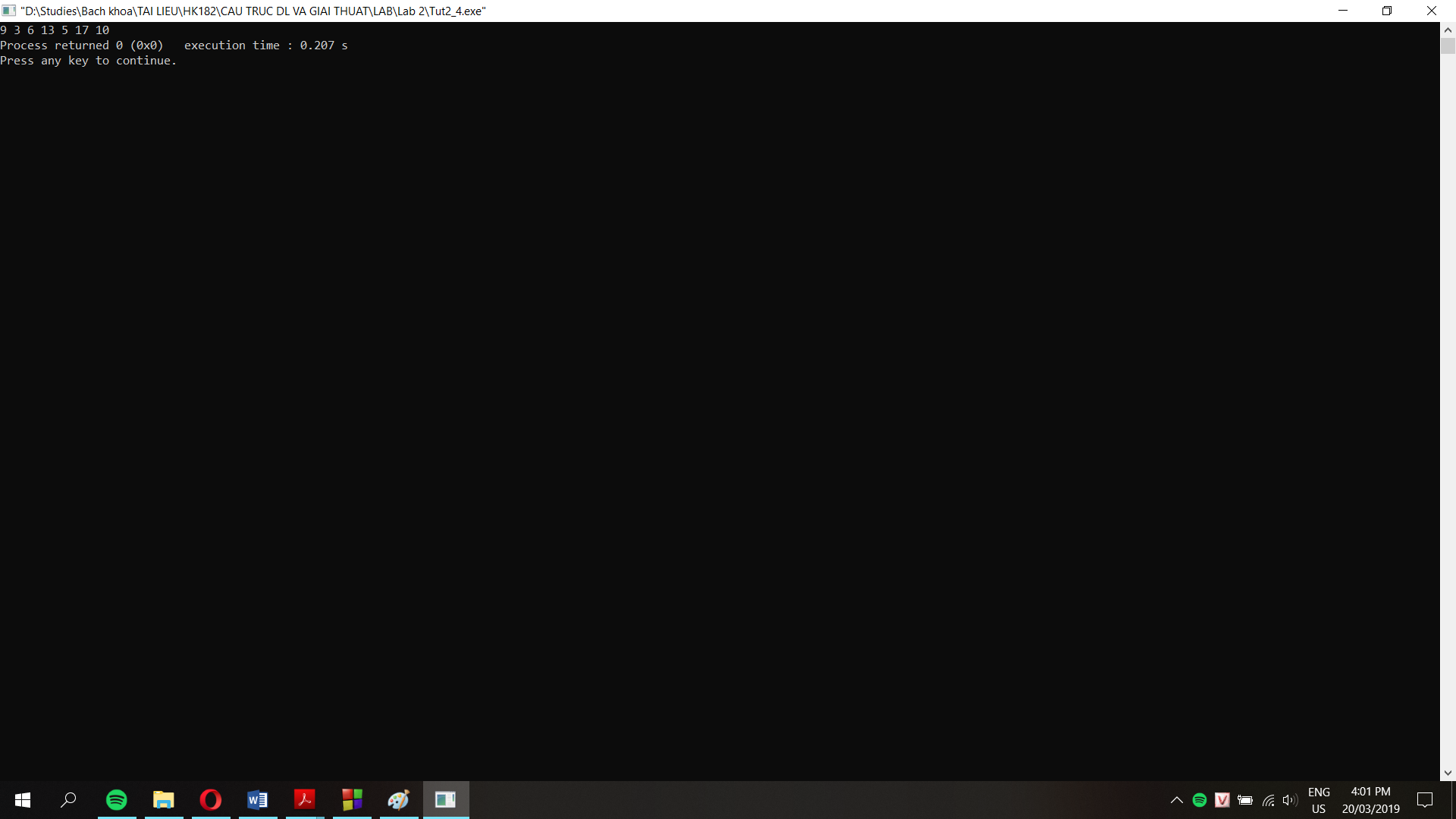
cout<<Head->data<<" ";

return (printNode(Head->next));

}

}

**Kết quả (sử dụng list của bài 4)**



**b.**

void func1(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

if (temp->next == NULL) {

temp->next = head;

return;

}

temp = temp->next;

}

}

Hàm func1() duyệt con trỏ temp qua tất cả các phần tử của linked list, khi gặp con trỏ cuối list (pTail) thì gán giá trị pTail->next = pHead. Khi này hàm in các phần tử của linked list ở câu a sẽ chạy vô hạn vì không gặp được phần tử cuối cùng của list có trường next = NULL.

**Question 6:**

node\* deleteNth(node\* head, int n){

node \*result;

int counter = countList(head); //using the function in Lab2, Question 2b, count the number of elements in a list

if (n==0){ //if n=0 delete the head and return a new list

result = head->next;

delete(head);

return result;

}

else if (n<0 || n>counter){ //if n<0 or greater than the length of the list, do nothing

return head;

}

else {

int counter2 = 0;

result = head;

node \*temp;

while (counter2 != n-1){ //point result to the n-1 node

result = result->next;

counter2 += 1;

}

temp = result;

result = result->next;

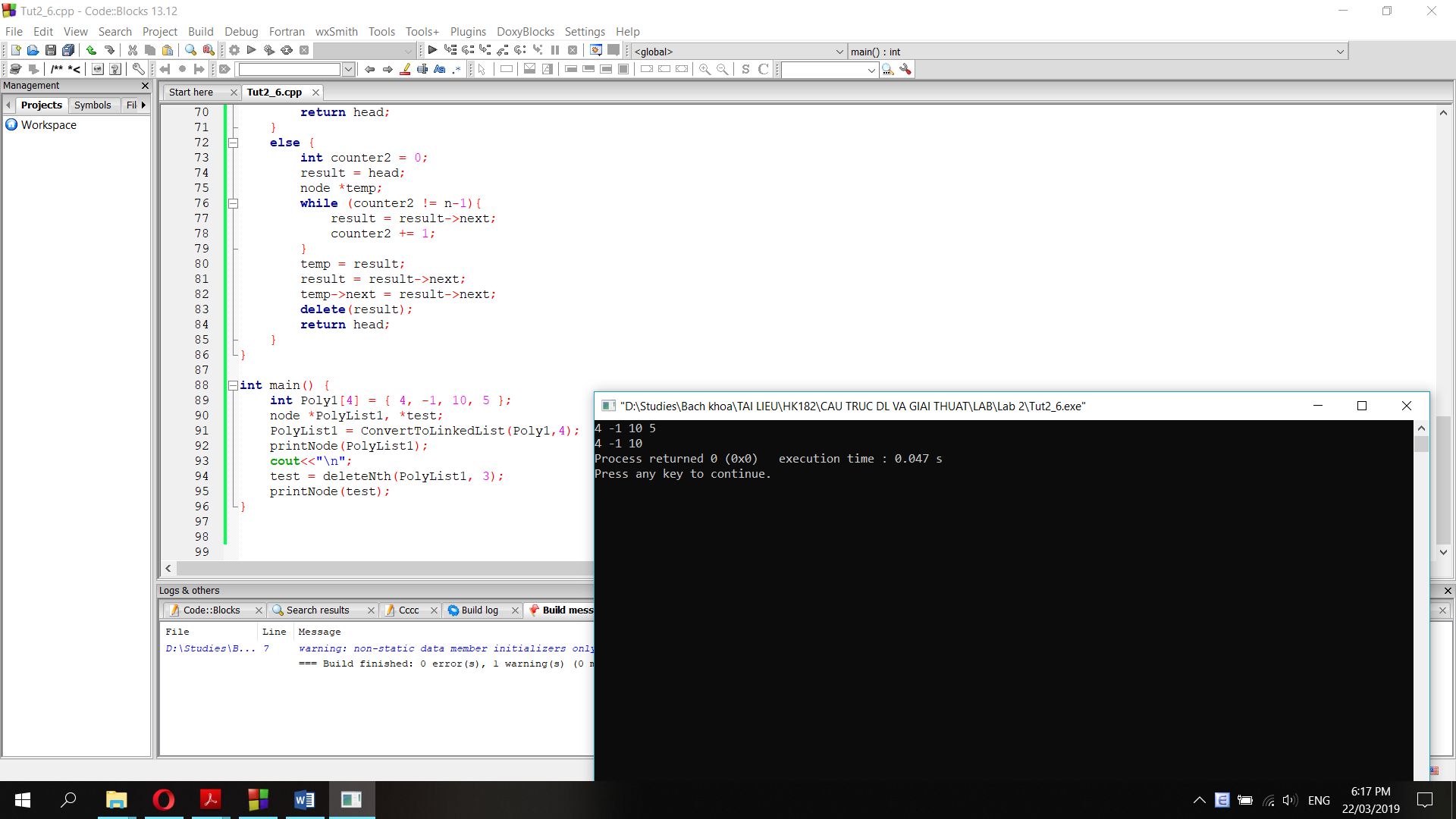
temp->next = result->next;

delete(result);

return head;

}

}



**LAB 2**

**Question 1: Function turns a list into a linked list**

node \*ConvertToLinkedList(int List[], int Size){

node \*pHead, \*pTemp, \*pTemp2;

if (Size==0){

return 0;

}

else if (Size==1){

pHead = new node;

pHead->data = List[0];

return pHead;

}

else {

int i=0;

pHead = new node;

pHead->data = List[0];

pTemp = new node;

pTemp->data = List[1];

pTemp->next = NULL;

pHead->next = pTemp;

for (i=2; i<Size; i++){

pTemp2 = new node;

pTemp->next = pTemp2;

pTemp2->data = List[i];

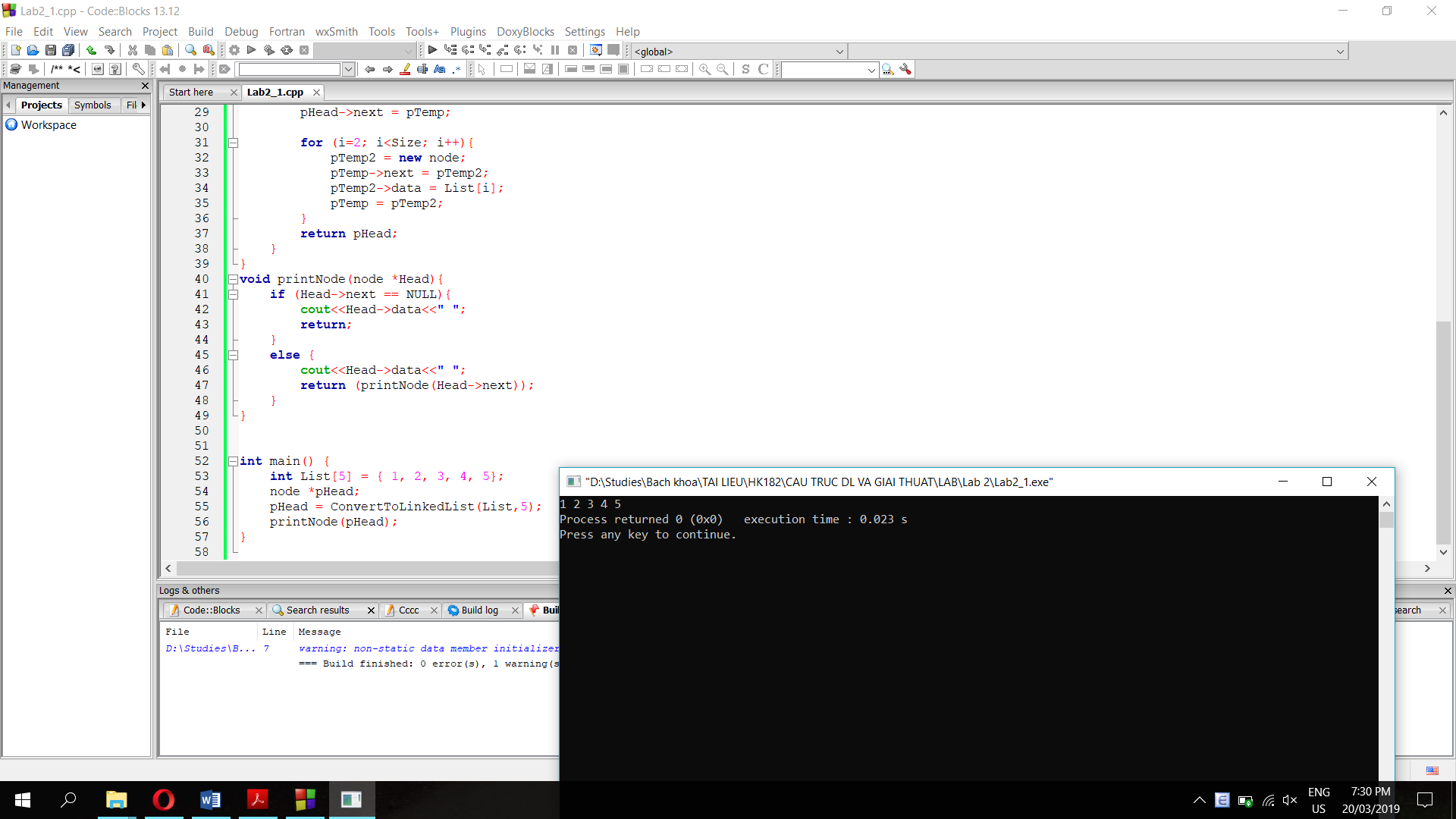
pTemp = pTemp2;

}

return pHead;

}

}



**Question 2:**

**a. A function traversal the linked list and print all data:**

struct node{

int data;

node\* next = NULL;

};

void printNode(node \*Head){

if (Head->next == NULL){

cout<<Head->data<<" ";

return;

}

else {

cout<<Head->data<<" ";

return (printNode(Head->next));

}

}

**Kết quả như câu 1.**

**b. A function sorts the linked list**

//count the number of elements in a linked list

int countList(node \*head){

int counter=1;

while (head->next != NULL){

counter += 1;

head = head->next;

}

return counter;

}

//traversal through the list and swap elements to sort list

void sortLinkedList(node \*head){

int counter;

counter = countList(head);

node \*temp1 = head;

node \*temp2 = head;

for (int i=0; i<counter; i++){

for (int j=0; j<counter - 1; j++){

if (temp1->data < temp2->data){

int temp = temp1->data;

temp1->data = temp2->data;

temp2->data = temp;

}

temp2 = temp2->next;

}

temp2 = head;

temp1 = head->next;

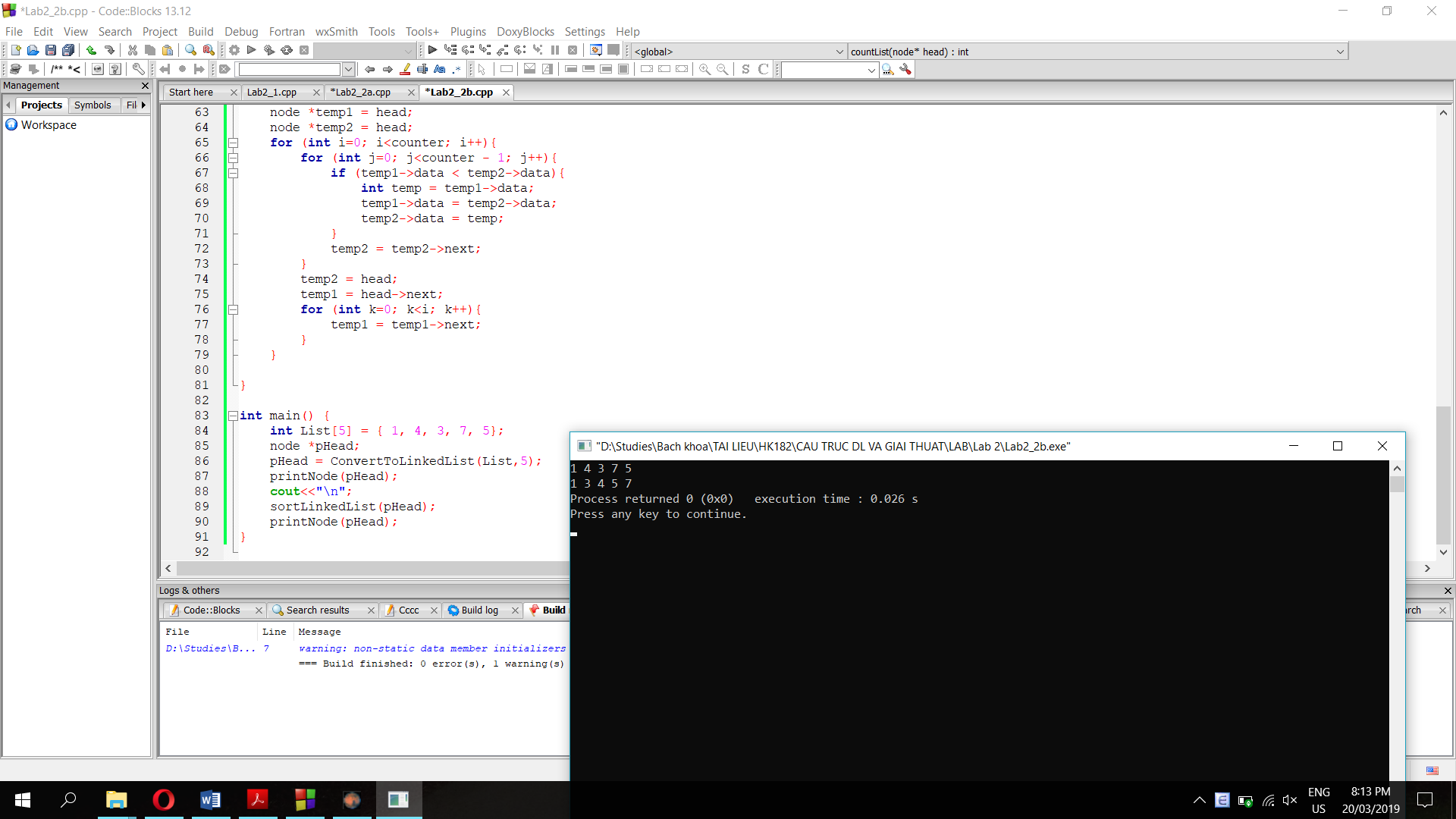
for (int k=0; k<i; k++){

temp1 = temp1->next;

}

}

}



**Câu 3: Function reverses a linked list:**

void reverseList(node \*&head){

node \*pHead;

node \*pTemp;

//empty list

if (head == NULL){

return;

}

pHead = head;

pTemp = head->next;

//list has 1 element

if (pTemp == NULL){

return;

}

reverseList(pTemp);

//switch 2 elements at one time and point the head to the last element of the original list

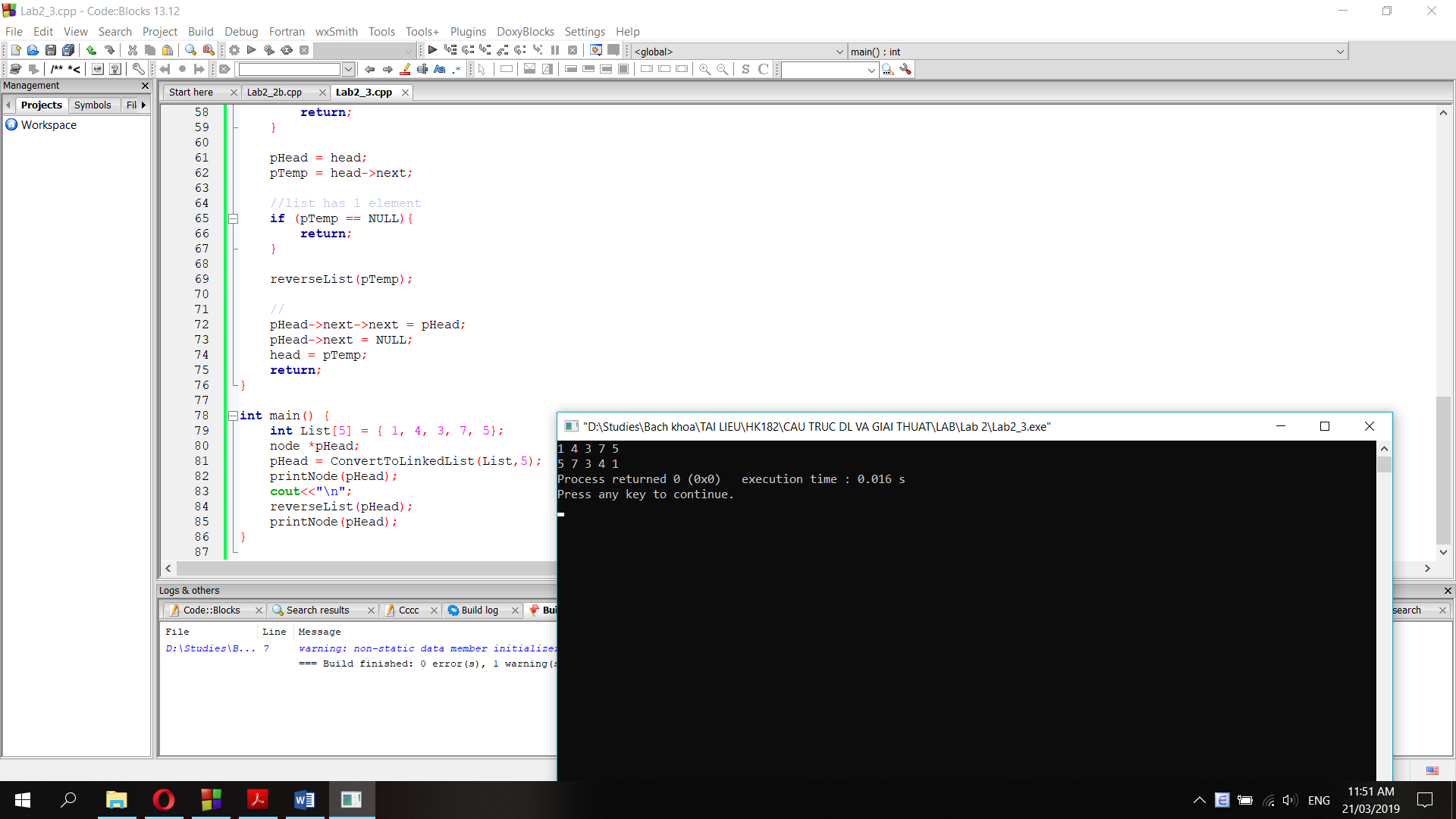
pHead->next->next = pHead;

pHead->next = NULL;

head = pTemp;

return;

}



**Question 4:**

**a. Function adds 2 Polynomials as linked lists:**

node \*addPolyList(node \*PolyList1, node \*PolyList2){

node \*result, \*head;

result = new node; //create a linked list as the result

head = result; //head is the head of the result list

if (PolyList1==NULL && PolyList2 != NULL){

return PolyList2; //in case we reach the end of list 1, return list 2

}

else if (PolyList1 != NULL && PolyList2 == NULL){

return PolyList1; //in case we reach the end of list 2, return list 1

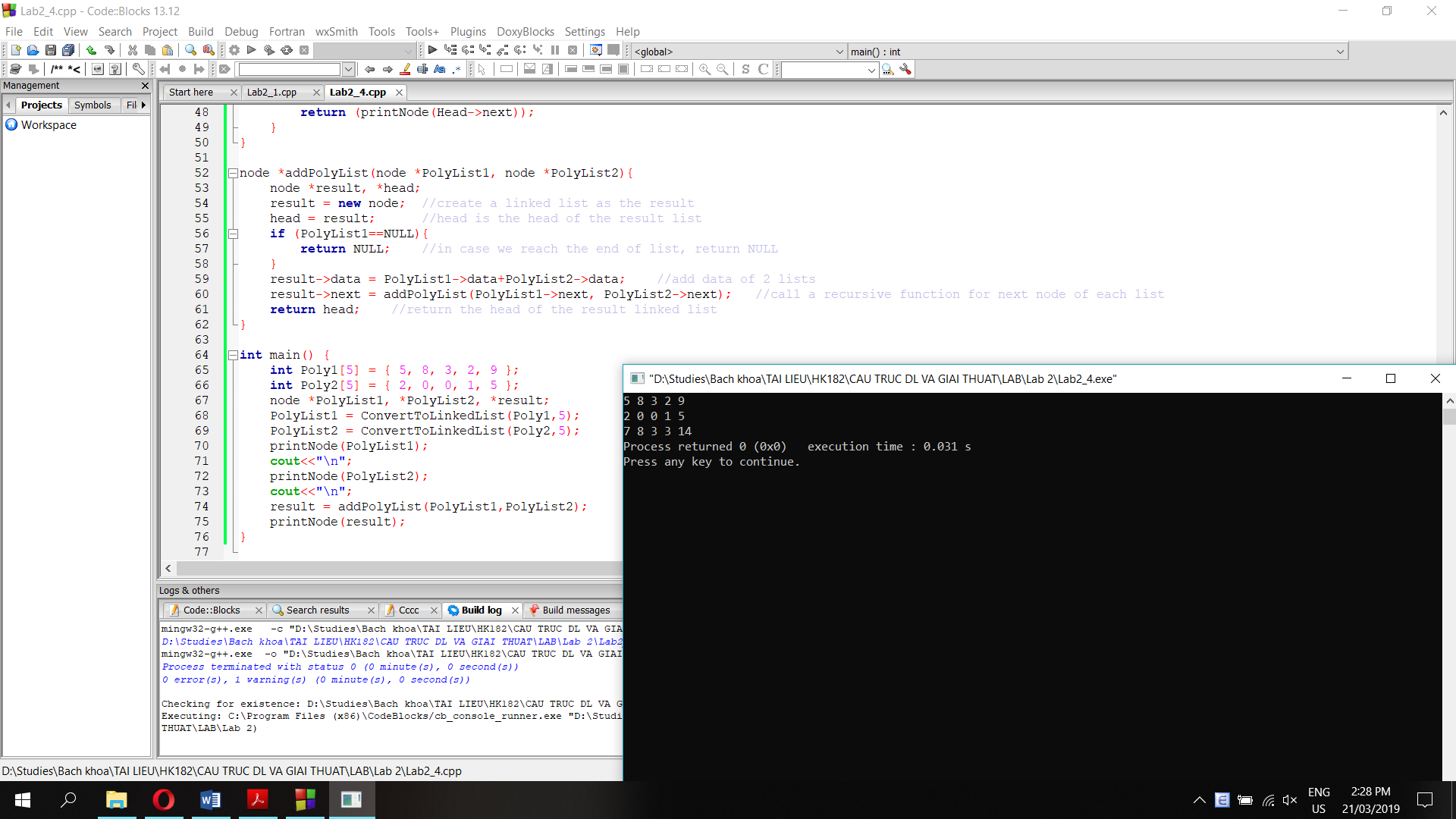
}

result->data = PolyList1->data+PolyList2->data; //add data of 2 lists

result->next = addPolyList(PolyList1->next, PolyList2->next); //call a recursive function for next node of each list

return head; //return the head of the result linked list

}



**b. Function sub 2 Polynomials as linked lists:**

node \*subPolyList(node \*PolyList1, node \*PolyList2){

node \*result, \*head;

result = new node; //create a linked list as the result

head = result; //head is the head of the result list

if (PolyList1==NULL && PolyList2 != NULL){

result->data = 0-PolyList2->data;

result->next = subPolyList(PolyList1, PolyList2->next);

} //in case we reach the end of list 1, return the negative of list 2

else if (PolyList1 != NULL && PolyList2 == NULL){

return PolyList1; //in case we reach the end of list 2, return list 1

}

else if (PolyList1 == NULL && PolyList2 == NULL){

return NULL;

}

else {

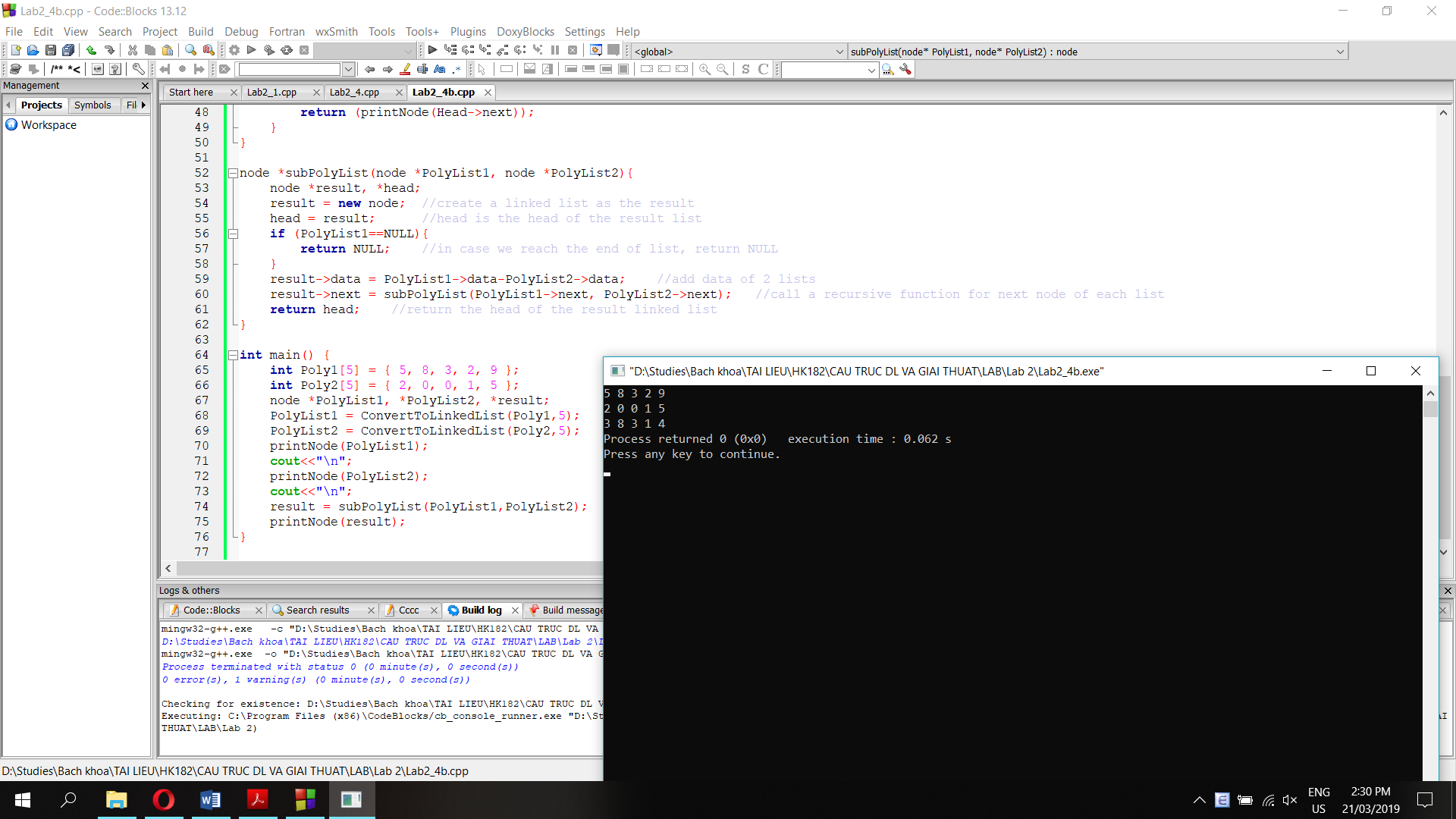
result->data = PolyList1->data-PolyList2->data; //add data of 2 lists

result->next = subPolyList(PolyList1->next, PolyList2->next); //call a recursive function for next node of each list

}

return head; //return the head of the result linked list

}



**Question 5: Function multiplies Polynomials as linked lists**

//Add 2 polynomial lists as in Question 4a

node \*addPolyList(node \*PolyList1, node \*PolyList2){

node \*result, \*head;

result = new node; //create a linked list as the result

head = result; //head is the head of the result list

if (PolyList1==NULL && PolyList2 != NULL){

return PolyList2; //in case we reach the end of list 1, return list 2

}

else if (PolyList1 != NULL && PolyList2 == NULL){

return PolyList1; //in case we reach the end of list 2, return list 1

}

result->data = PolyList1->data+PolyList2->data; //add data of 2 lists

result->next = addPolyList(PolyList1->next, PolyList2->next); //call a recursive function for next node of each list

return head; //return the head of the result linked list

}

//multiply each element of polynomial 1 with one element of polynomial 2

node \*mulEachElementPolyList(node \*PolyList1, node \*PolyList2){

node \*result, \*head;

result = new node;

head = result;

if (PolyList1==NULL){

return NULL;

}

result->data = PolyList1->data \* PolyList2->data;

result->next = mulEachElementPolyList(PolyList1->next, PolyList2);

return head;

}

//shift right each element of the linked list n spots

node \*shift(node \*PolyList, int n){

node \*head;

head = new node;

if (n==0){

return PolyList;

}

head->data = 0;

head->next = shift(PolyList, n-1);

}

//multiplies 2 polynomials using 3 functions above

node \*mulPolyList(node \*PolyList1, node \*PolyList2){

node \*result1, \*result2, \*result, \*head;

int counter = countList(PolyList2);

int n = 0;

if (n == counter){

return NULL; //in case we reach the end of the list

}

result = NULL;

while (PolyList2 != NULL){

//multiply list 1 with each element of list 2, then add the result to result list (shift left 1 spot each time)

result = addPolyList(result, shift(mulEachElementPolyList(PolyList1, PolyList2), n));

n+=1;

PolyList2 = PolyList2->next; //move to the next element of list 2 and return the loop

}

return result;

}

