**Implementation of basic data structures**

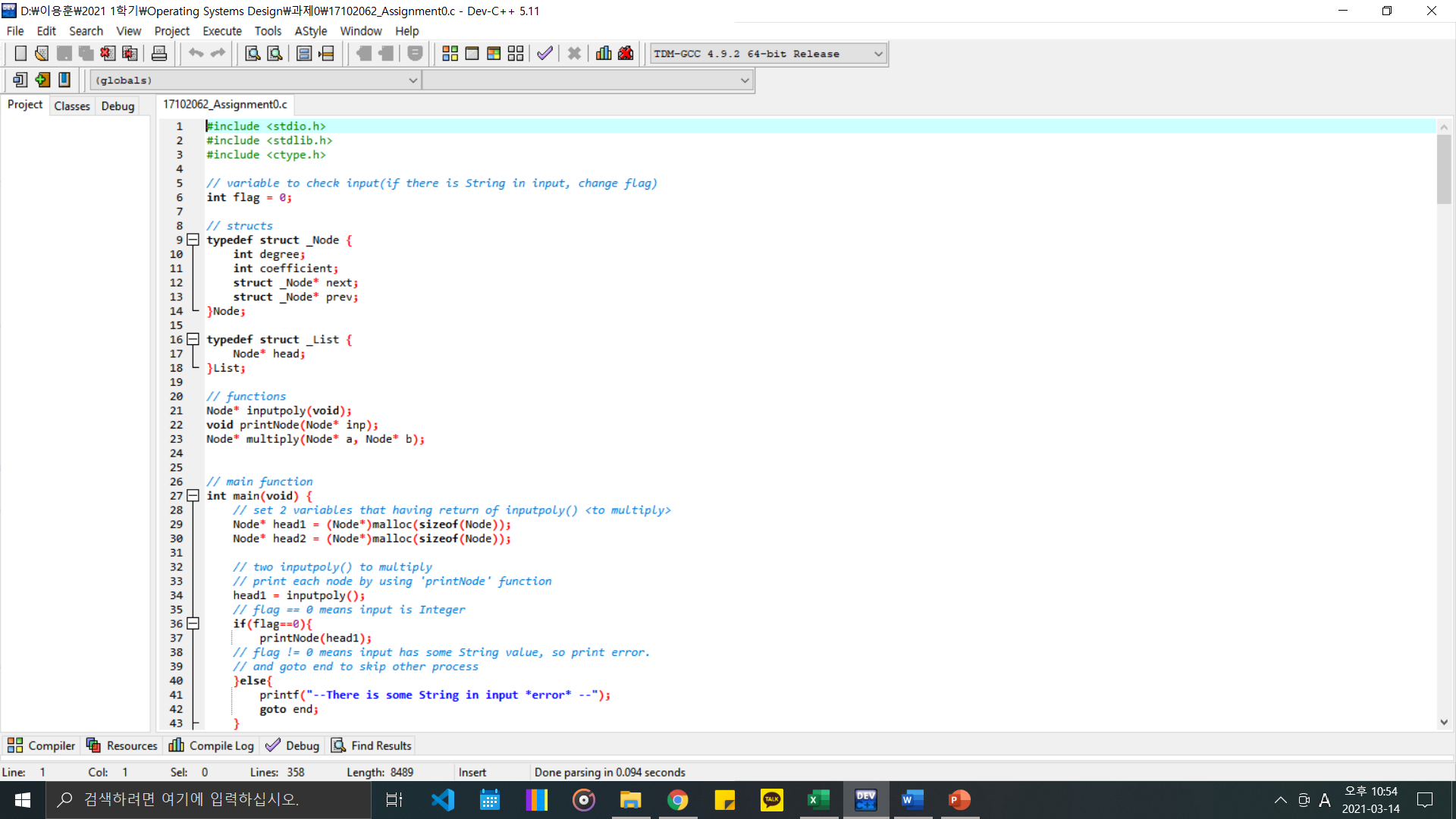
**Subject = Operating Systems Design**

**Major = ITM**

**Student ID = 17102062**

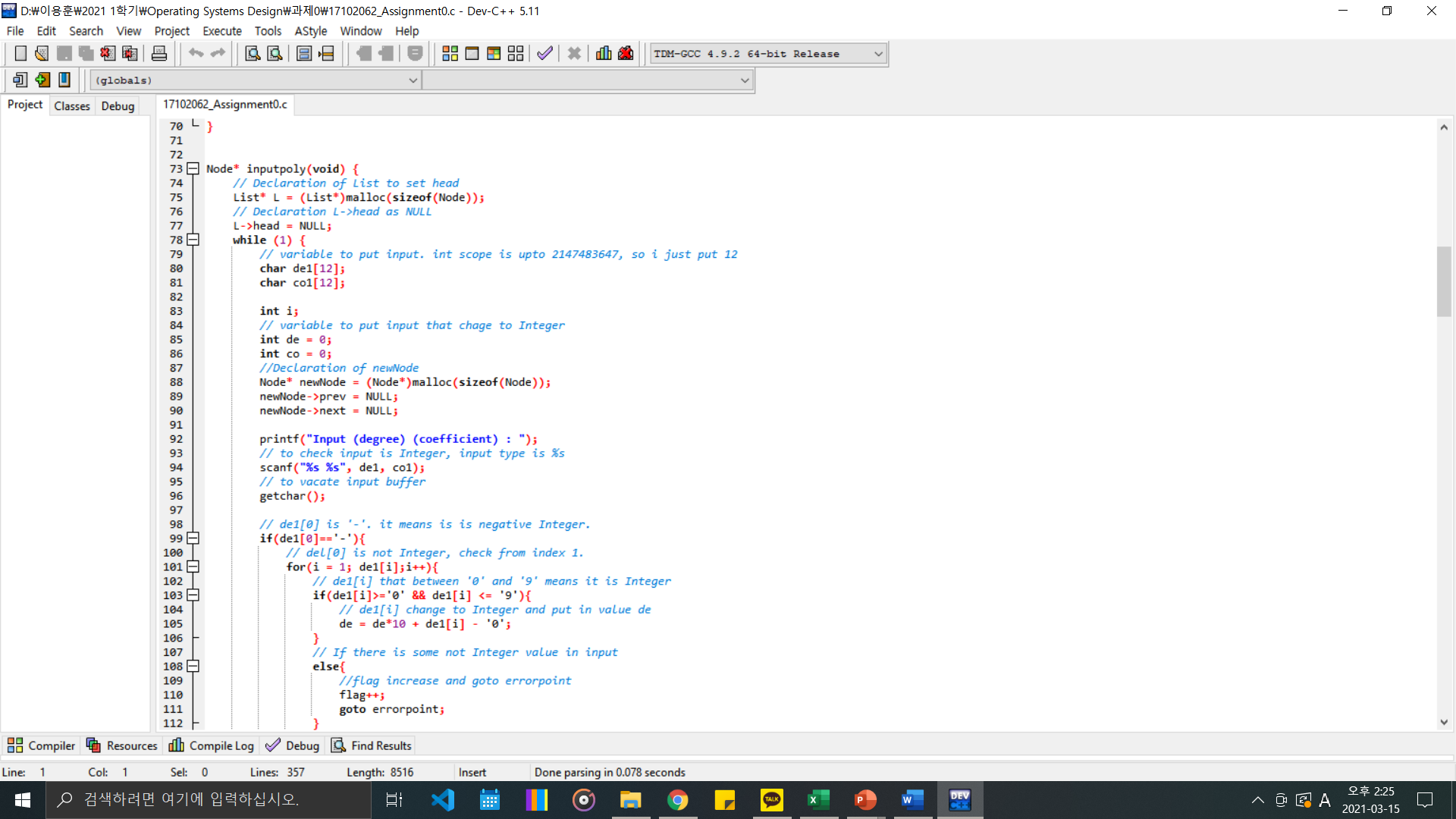
**Name = Lee Yong Hoon(이용훈)**

**A. structure**

****

**There is List structure to use head**

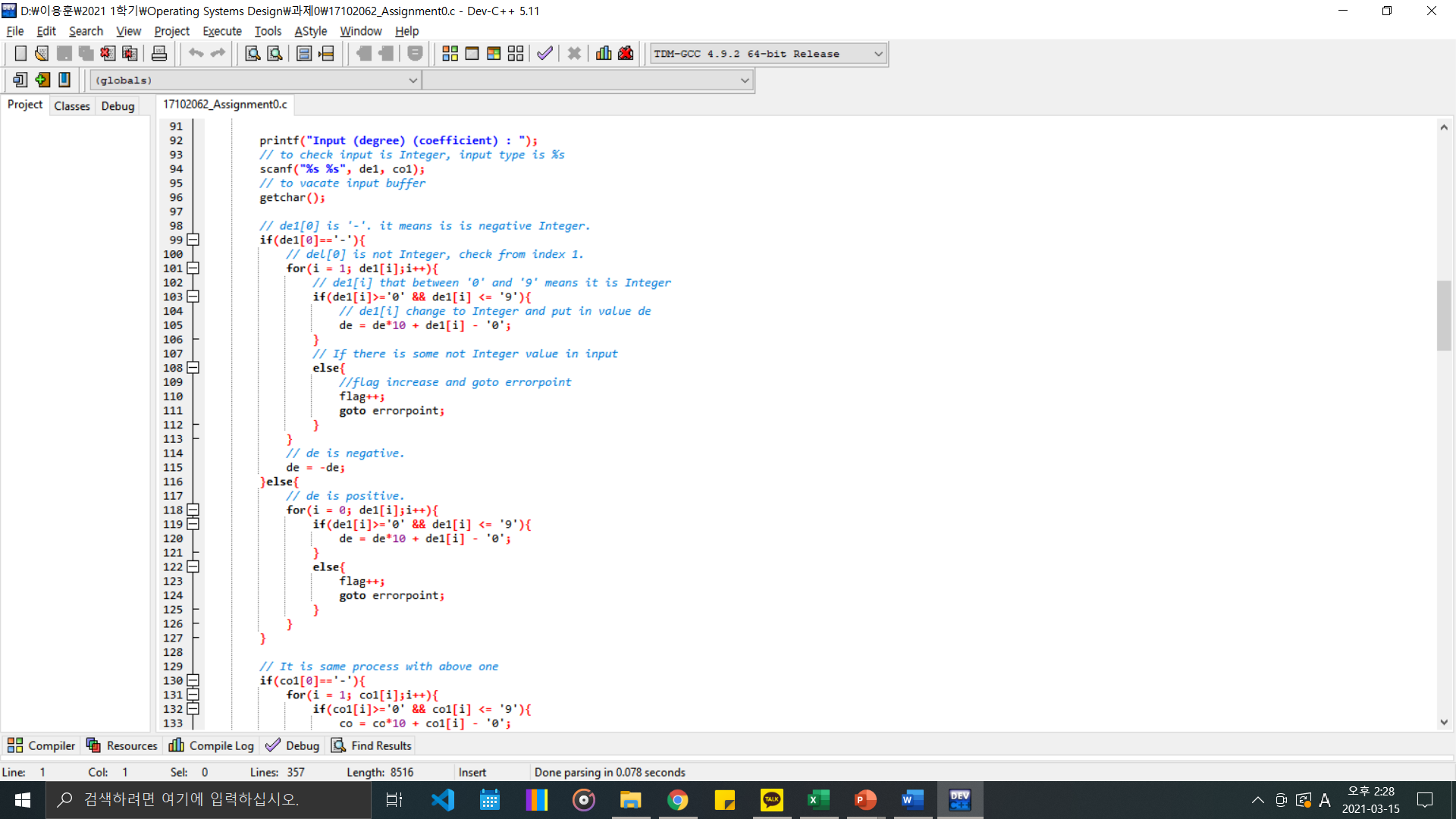
**B. ‘inputpoly’ function**

****

**To check whether input is Integer or not, input type is char.**

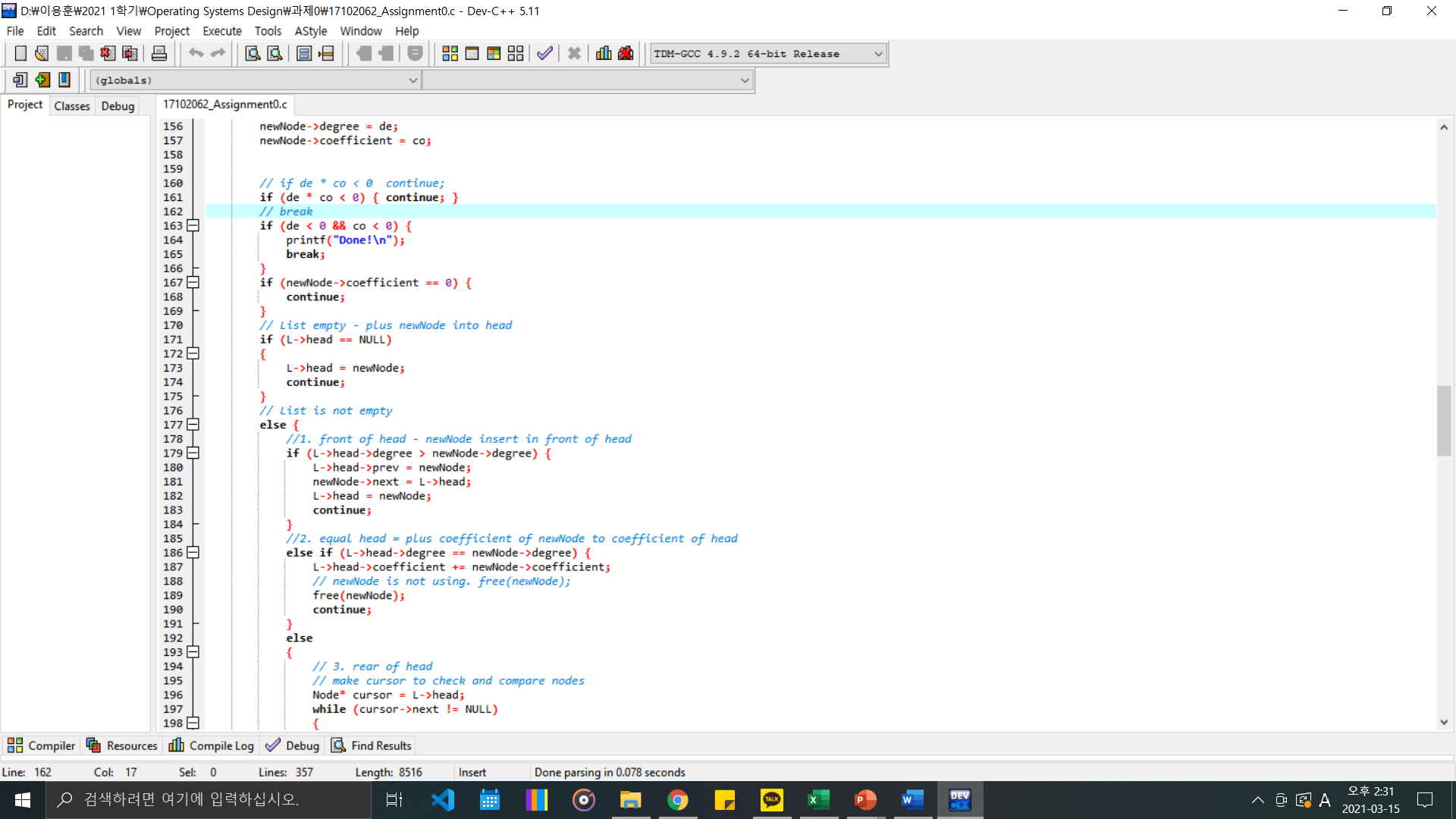
**There is two variable ‘de1’ and ‘co1’ to have input value.**

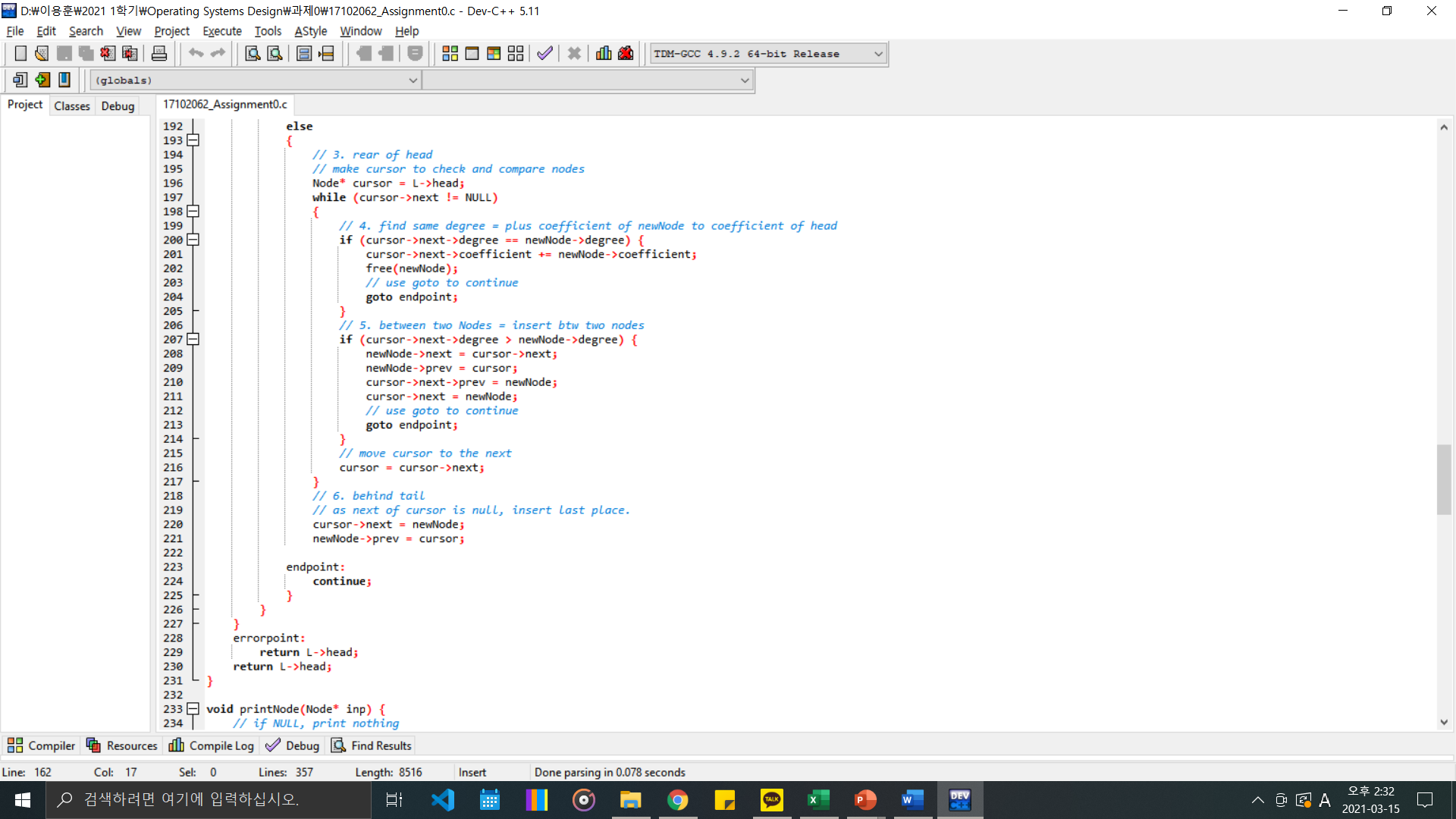
**There is two variable ‘de’ and ‘co’ to have Integer value. Code check ‘de1’ and ‘de2’ and find integer value and put that value into ‘de’ and ‘co’ variable.**

****

**We check ‘del1’ is integer value or not. By using 105 line, we can change ‘de1[I]’ value to int ‘de’ value.**

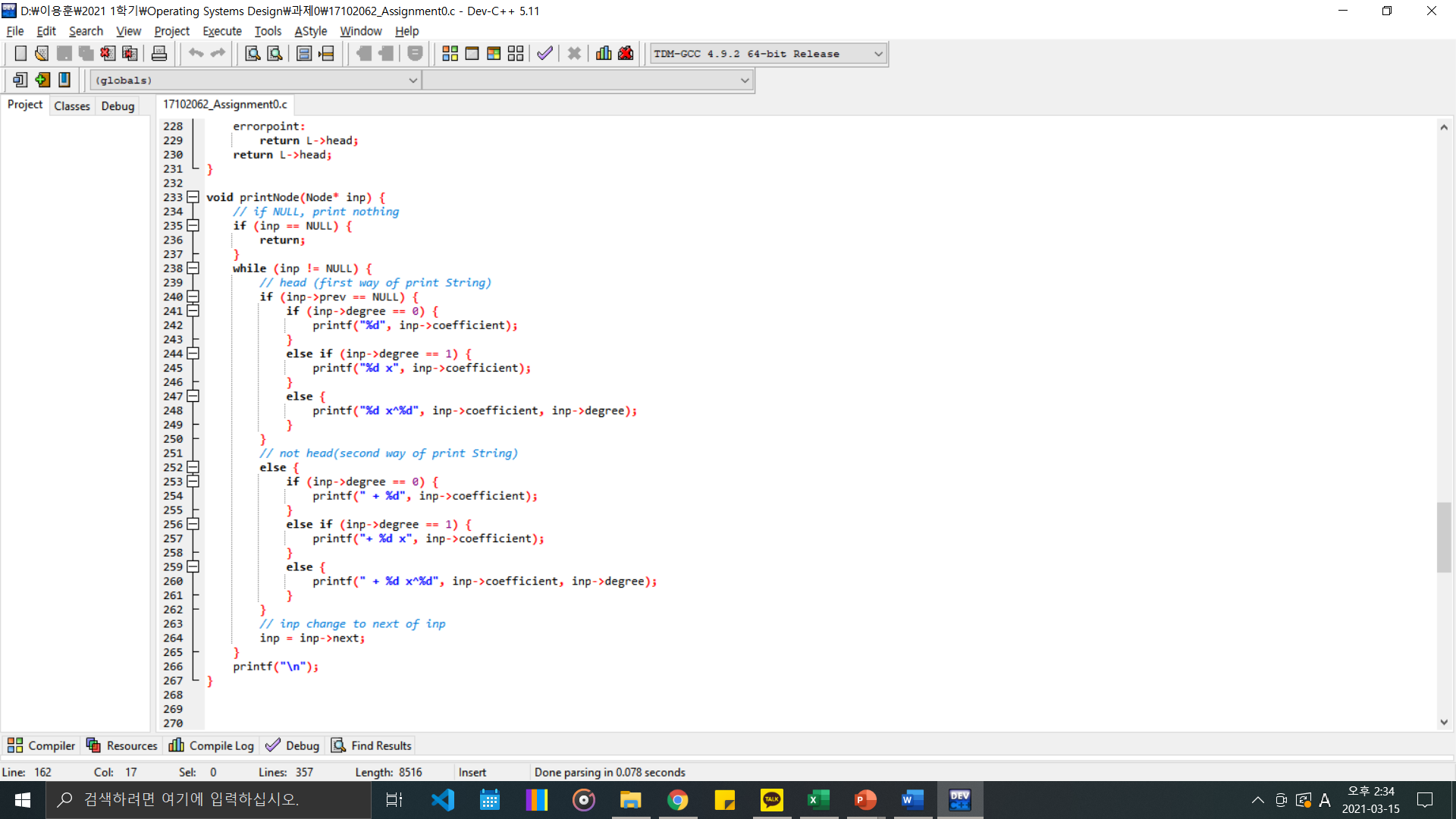
**To check whether it is positive or negative, there is two case that ‘de1[0]’ is ‘-‘ or not.**

****

****

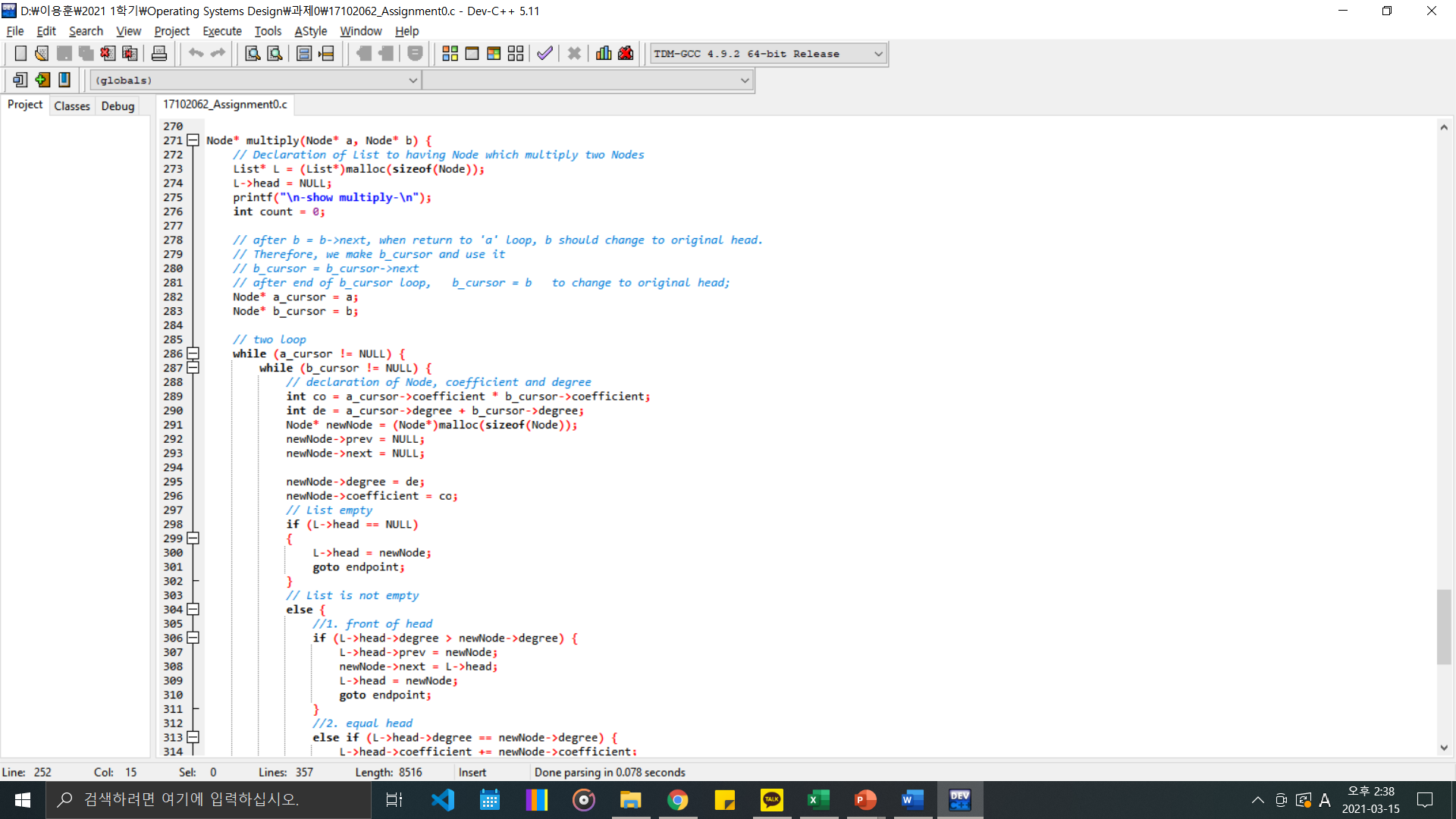
**To make doubly linked list, we divided many case and according to relationship with nodes, we put node properly.**

**C. ‘printNode’ function**

****

**Parameter is Node \* inp. If int is null, return 0. To make output properly we divide two case. Is node is head or not. We make some case that if degree is 0 or not.**

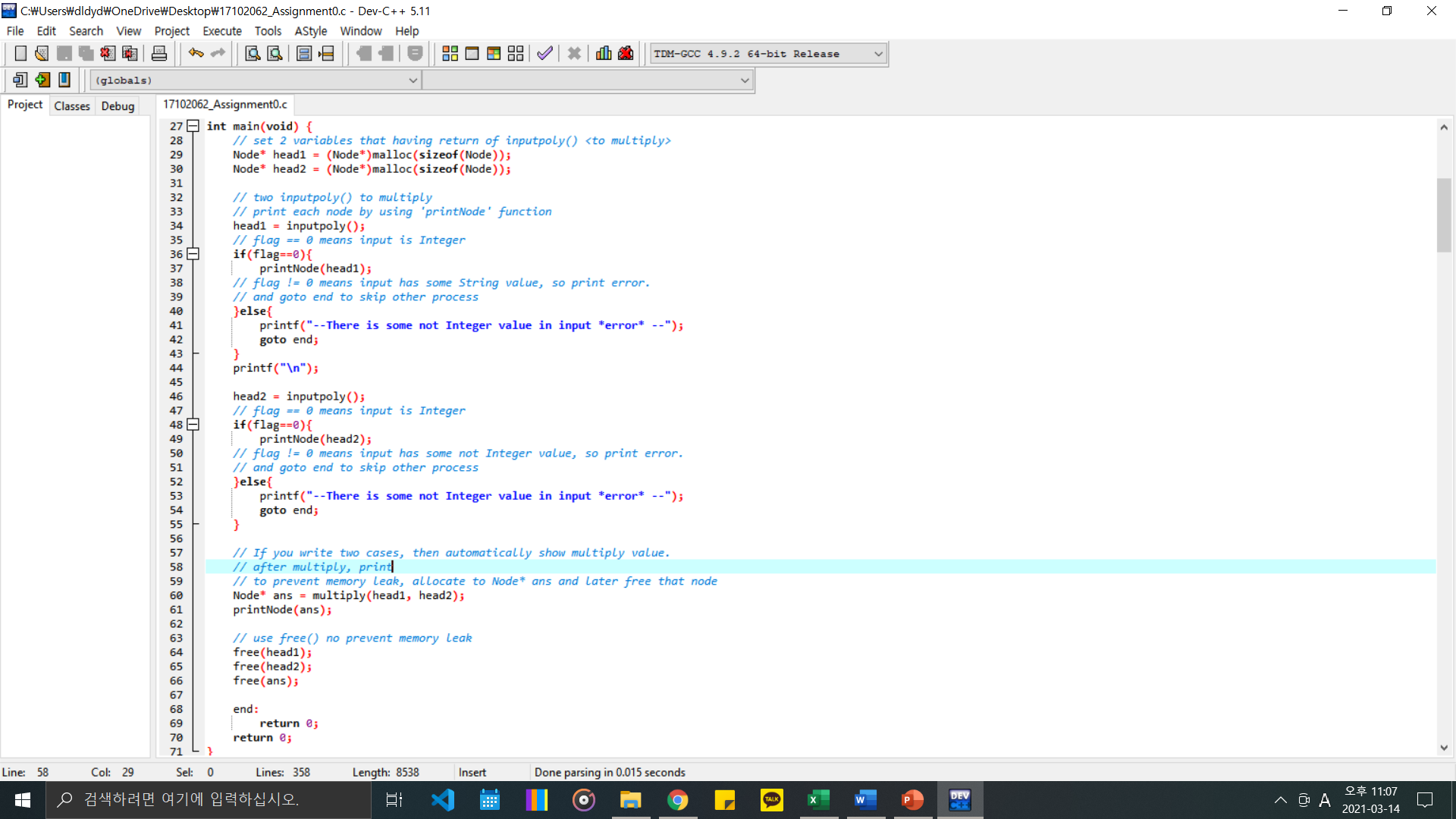
**D. ‘multiply’ function**

****

**Parameters are node\*a and node\*b.**

**We make result list L to take multiply node and two cursor to check two doubly linked list. While using double while loop, we multiply two nodes and put the value into result list L. Input new node into result list L is almost same with inputpoly code so you can see c code file more detail.**

**E. main function**

****

**There are two inputpoly function. And if you execute two inputpoly then automatically show you multiply results.**

**To check whether input is Integer, we use global variable ‘flag’ and if there is some none Integer value in input, print “--There is some not Integer value in input \*error\* --” and goto end.**

**F. execution**

**1. normal case**

**If you write two input, then automatically show multiply result.**

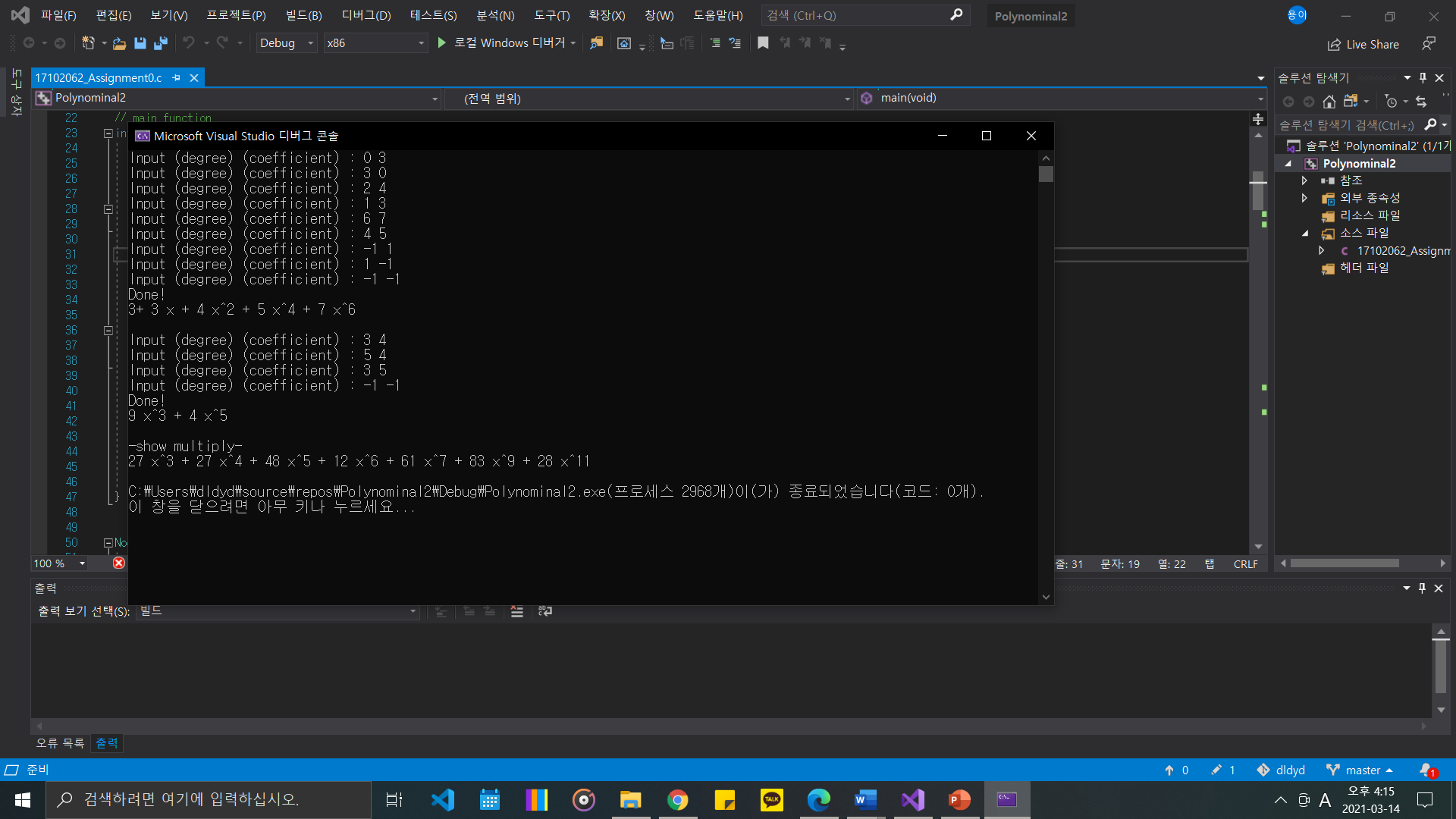
**All exception is applied**

**0 3 is null**

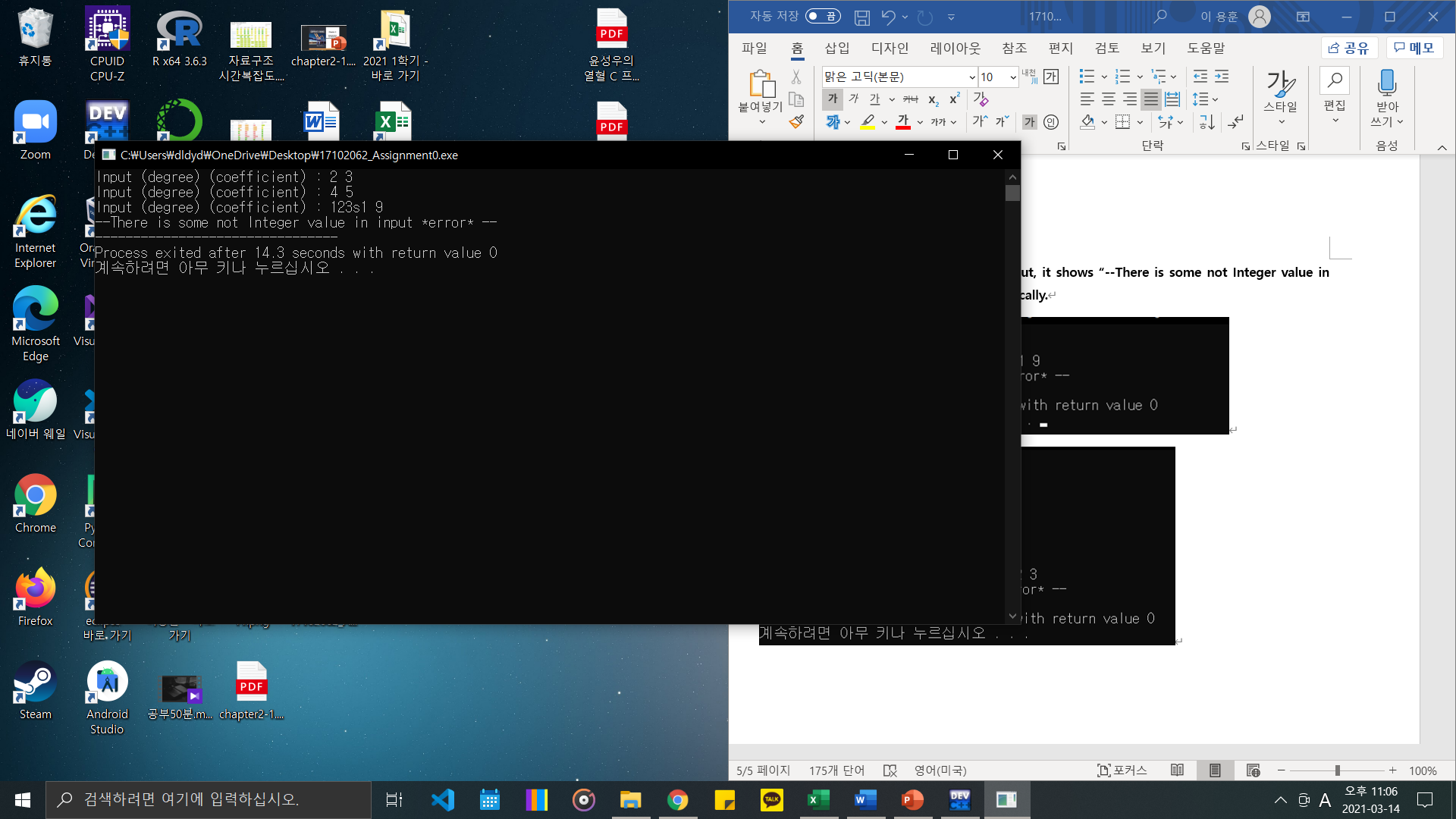
**3 0 is only degree value**

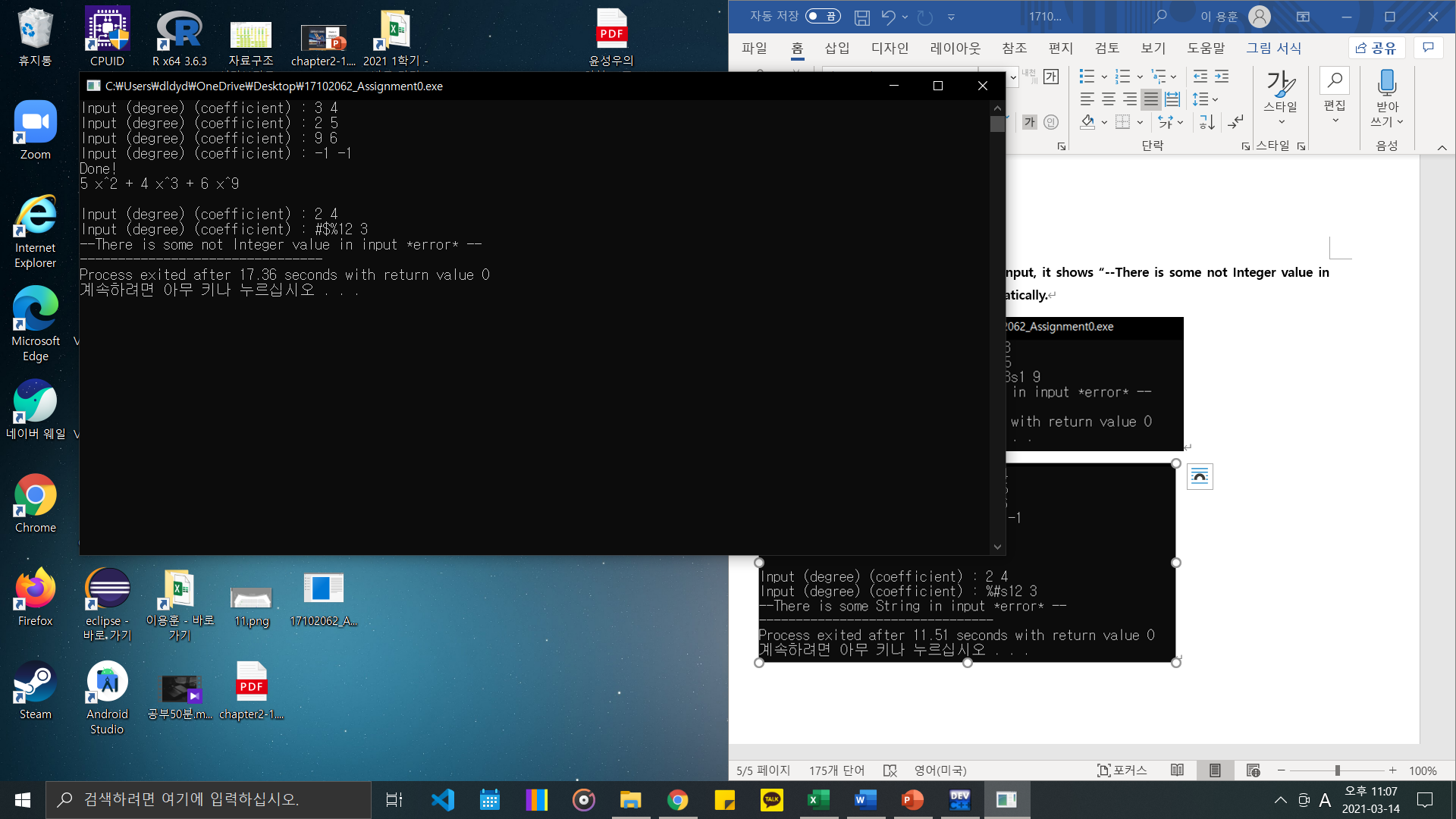
**-1 1 and 1 -1 is nothing**

**-1 -1 is end process**



**2. If there is some other type value in input, it shows “--There is some not Integer value in input \*error\* --” and end process automatically.**

****



**G. Code**

#include <stdio.h>

#include <stdlib.h>

// variable to check input(if there is not Integer value in input, change flag)

int flag = 0;

// structs

typedef struct \_Node {

int degree;

int coefficient;

struct \_Node\* next;

struct \_Node\* prev;

}Node;

typedef struct \_List {

Node\* head;

}List;

// functions

Node\* inputpoly(void);

void printNode(Node\* inp);

Node\* multiply(Node\* a, Node\* b);

// main function

int main(void) {

// set 2 variables that having return of inputpoly() <to multiply>

Node\* head1 = (Node\*)malloc(sizeof(Node));

Node\* head2 = (Node\*)malloc(sizeof(Node));

// two inputpoly() to multiply

// print each node by using 'printNode' function

head1 = inputpoly();

// flag == 0 means input is Integer

if(flag==0){

printNode(head1);

// flag != 0 means input has some String value, so print error.

// and goto end to skip other process

}else{

printf("--There is some not Integer value in input \*error\* --");

goto end;

}

printf("\n");

head2 = inputpoly();

// flag == 0 means input is Integer

if(flag==0){

printNode(head2);

// flag != 0 means input has some not Integer value, so print error.

// and goto end to skip other process

}else{

printf("--There is some not Integer value in input \*error\* --");

goto end;

}

// If you write two cases, then automatically show multiply value.

// after multiply, print

// to prevent memory leak, allocate to Node\* ans and later free that node

Node\* ans = multiply(head1, head2);

printNode(ans);

// use free() no prevent memory leak

free(head1);

free(head2);

free(ans);

end:

return 0;

return 0;

}

Node\* inputpoly(void) {

// Declaration of List to set head

List\* L = (List\*)malloc(sizeof(Node));

// Declaration L->head as NULL

L->head = NULL;

while (1) {

// variable to put input. int scope is upto 2147483647, so i just put 12

char de1[12];

char co1[12];

int i;

// variable to put input that chage to Integer

int de = 0;

int co = 0;

//Declaration of newNode

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->prev = NULL;

newNode->next = NULL;

printf("Input (degree) (coefficient) : ");

// to check input is Integer, input type is %s

scanf("%s %s", de1, co1);

// to vacate input buffer

getchar();

// de1[0] is '-'. it means is is negative Integer.

if(de1[0]=='-'){

// del[0] is not Integer, check from index 1.

for(i = 1; de1[i];i++){

// de1[i] that between '0' and '9' means it is Integer

if(de1[i]>='0' && de1[i] <= '9'){

// de1[i] change to Integer and put in value de

de = de\*10 + de1[i] - '0';

}

// If there is some not Integer value in input

else{

//flag increase and goto errorpoint

flag++;

goto errorpoint;

}

}

// de is negative.

de = -de;

}else{

// de is positive.

for(i = 0; de1[i];i++){

if(de1[i]>='0' && de1[i] <= '9'){

de = de\*10 + de1[i] - '0';

}

else{

flag++;

goto errorpoint;

}

}

}

// It is same process with above one

if(co1[0]=='-'){

for(i = 1; co1[i];i++){

if(co1[i]>='0' && co1[i] <= '9'){

co = co\*10 + co1[i] - '0';

}

else{

flag++;

goto errorpoint;

}

}

co = -co;

}else{

for(i = 0; de1[i];i++){

if(co1[i]>='0' && co1[i] <= '9'){

co = co\*10 + co1[i] - '0';

}

else{

if(co1[i]==0){

break;

}

flag++;

goto errorpoint;

}

}

}

newNode->degree = de;

newNode->coefficient = co;

// if de \* co < 0 continue;

if (de \* co < 0) { continue; }

// break

if (de < 0 && co < 0) {

printf("Done!\n");

break;

}

if (newNode->coefficient == 0) {

continue;

}

// List empty - plus newNode into head

if (L->head == NULL)

{

L->head = newNode;

continue;

}

// List is not empty

else {

//1. front of head - newNode insert in front of head

if (L->head->degree > newNode->degree) {

L->head->prev = newNode;

newNode->next = L->head;

L->head = newNode;

continue;

}

//2. equal head = plus coefficient of newNode to coefficient of head

else if (L->head->degree == newNode->degree) {

L->head->coefficient += newNode->coefficient;

// newNode is not using. free(newNode);

free(newNode);

continue;

}

else

{

// 3. rear of head

// make cursor to check and compare nodes

Node\* cursor = L->head;

while (cursor->next != NULL)

{

// 4. find same degree = plus coefficient of newNode to coefficient of head

if (cursor->next->degree == newNode->degree) {

cursor->next->coefficient += newNode->coefficient;

free(newNode);

// use goto to continue

goto endpoint;

}

// 5. between two Nodes = insert btw two nodes

if (cursor->next->degree > newNode->degree) {

newNode->next = cursor->next;

newNode->prev = cursor;

cursor->next->prev = newNode;

cursor->next = newNode;

// use goto to continue

goto endpoint;

}

// move cursor to the next

cursor = cursor->next;

}

// 6. behind tail

// as next of cursor is null, insert last place.

cursor->next = newNode;

newNode->prev = cursor;

endpoint:

continue;

}

}

}

errorpoint:

return L->head;

return L->head;

}

void printNode(Node\* inp) {

// if NULL, print nothing

if (inp == NULL) {

return;

}

while (inp != NULL) {

// head (first way of print String)

if (inp->prev == NULL) {

if (inp->degree == 0) {

printf("%d", inp->coefficient);

}

else if (inp->degree == 1) {

printf("%d x", inp->coefficient);

}

else {

printf("%d x^%d", inp->coefficient, inp->degree);

}

}

// not head(second way of print String)

else {

if (inp->degree == 0) {

printf(" + %d", inp->coefficient);

}

else if (inp->degree == 1) {

printf("+ %d x", inp->coefficient);

}

else {

printf(" + %d x^%d", inp->coefficient, inp->degree);

}

}

// inp change to next of inp

inp = inp->next;

}

printf("\n");

}

Node\* multiply(Node\* a, Node\* b) {

// Declaration of List to having Node which multiply two Nodes

List\* L = (List\*)malloc(sizeof(Node));

L->head = NULL;

printf("\n-show multiply-\n");

int count = 0;

// after b = b->next, when return to 'a' loop, b should change to original head.

// Therefore, we make b\_cursor and use it

// b\_cursor = b\_cursor->next

// after end of b\_cursor loop, b\_cursor = b to change to original head;

Node\* a\_cursor = a;

Node\* b\_cursor = b;

// two loop

while (a\_cursor != NULL) {

while (b\_cursor != NULL) {

// declaration of Node, coefficient and degree

int co = a\_cursor->coefficient \* b\_cursor->coefficient;

int de = a\_cursor->degree + b\_cursor->degree;

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->prev = NULL;

newNode->next = NULL;

newNode->degree = de;

newNode->coefficient = co;

// List empty

if (L->head == NULL)

{

L->head = newNode;

goto endpoint;

}

// List is not empty

else {

//1. front of head

if (L->head->degree > newNode->degree) {

L->head->prev = newNode;

newNode->next = L->head;

L->head = newNode;

goto endpoint;

}

//2. equal head

else if (L->head->degree == newNode->degree) {

L->head->coefficient += newNode->coefficient;

free(newNode);

goto endpoint;

}

else

{

//3. rear of head

Node\* cursor = L->head;

while (cursor->next != NULL)

{

// 4. find same degree

if (cursor->next->degree == newNode->degree) {

cursor->next->coefficient += newNode->coefficient;

free(newNode);

goto endpoint;

}

// 5. between two Nodes

if (cursor->next->degree > newNode->degree) {

newNode->next = cursor->next;

newNode->prev = cursor;

cursor->next->prev = newNode;

cursor->next = newNode;

goto endpoint;

}

cursor = cursor->next;

}

// 6. behind tail

cursor->next = newNode;

newNode->prev = cursor;

}

}

endpoint:

b\_cursor = b\_cursor->next;

count++;

}

// back to original b

b\_cursor = b;

a\_cursor = a\_cursor->next;

}

return L->head;

}