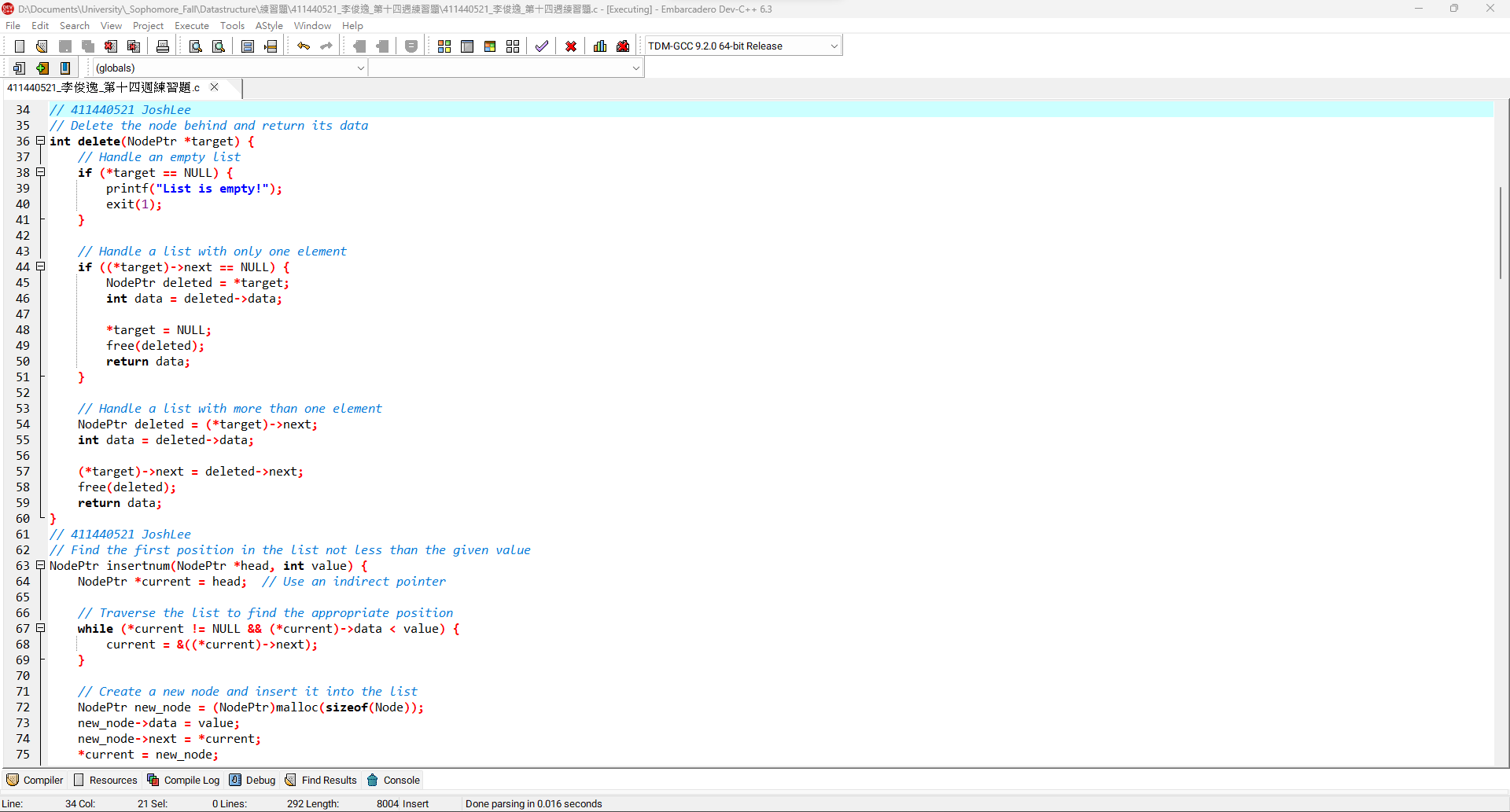
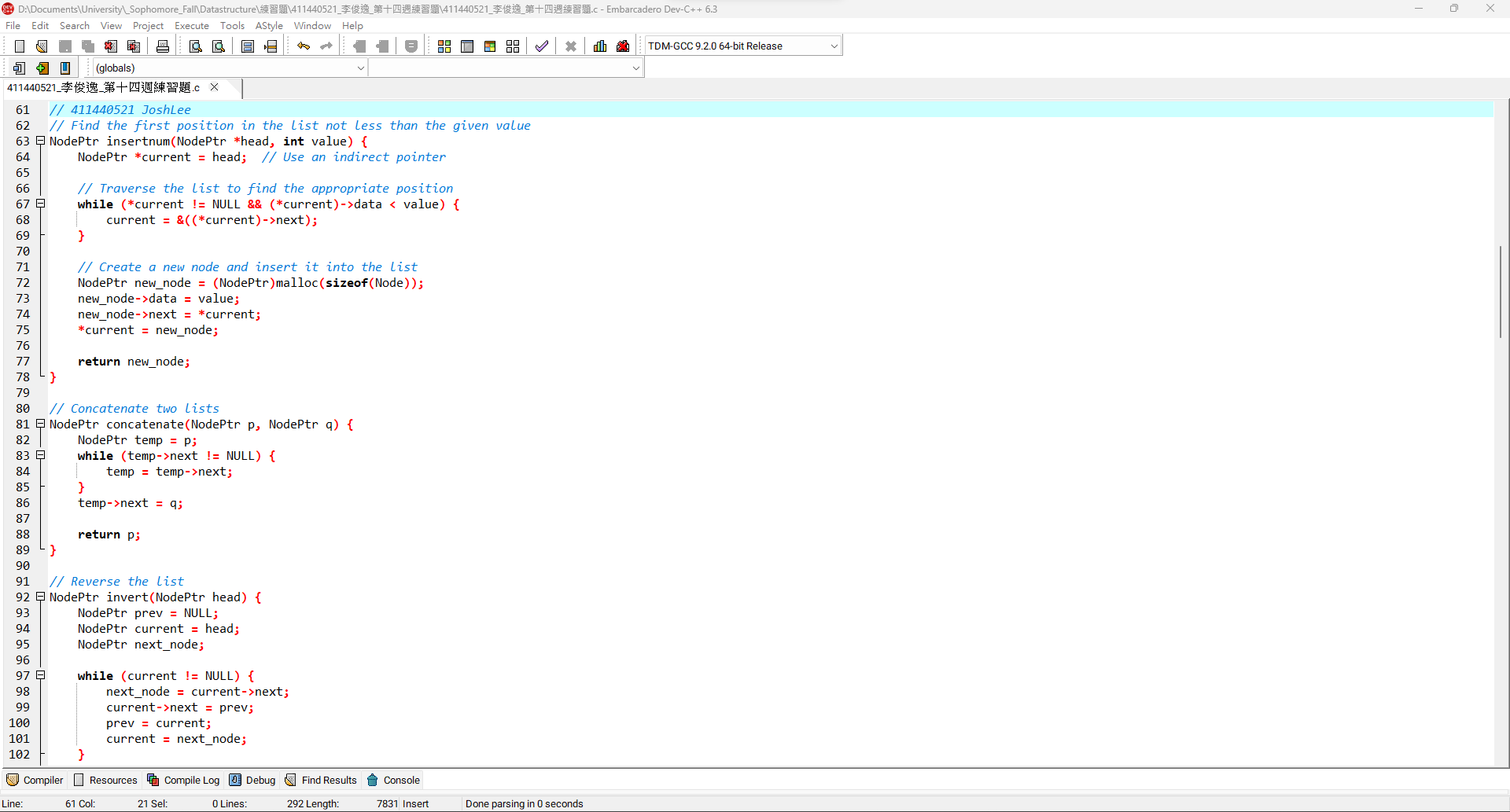
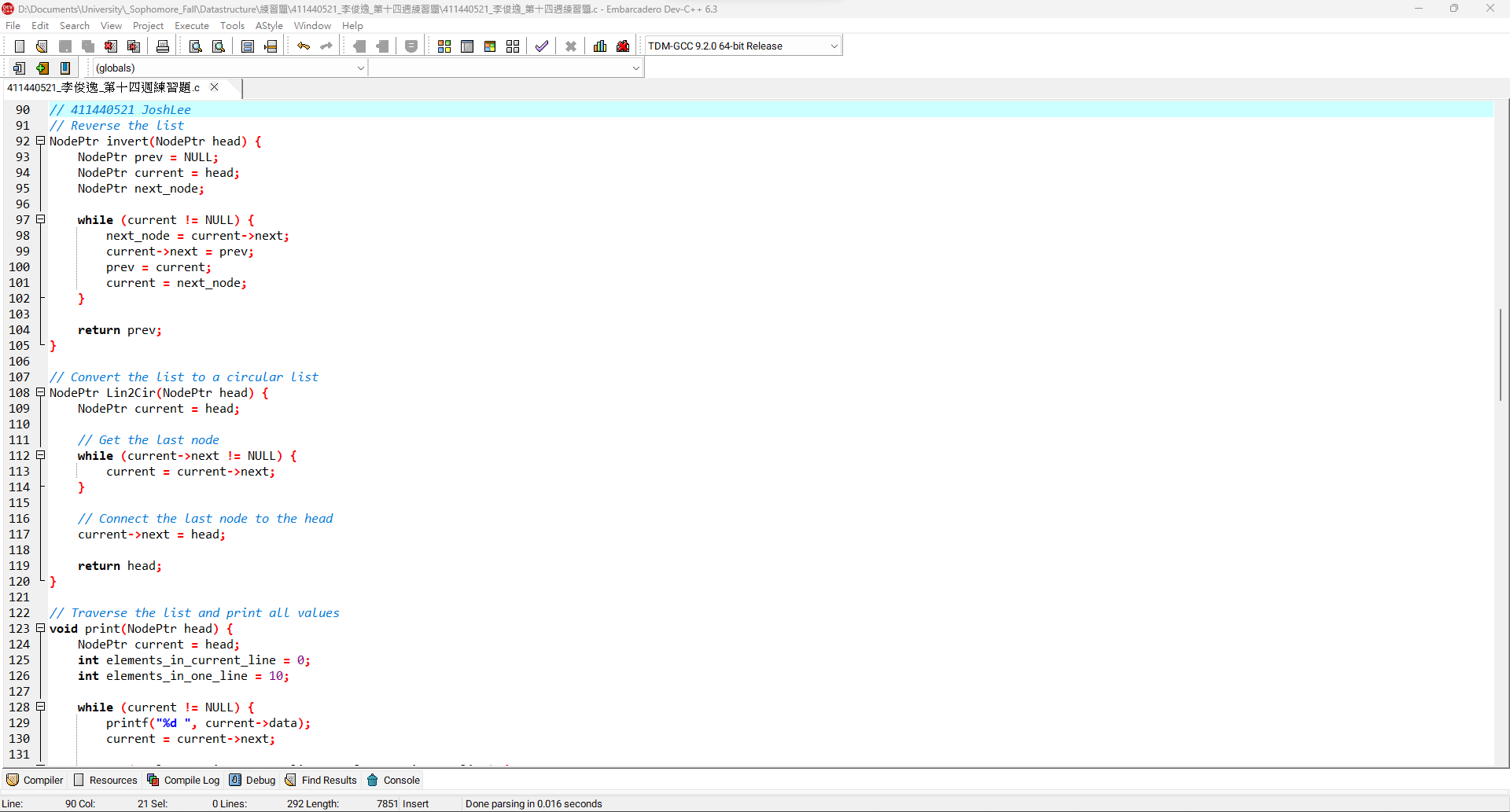
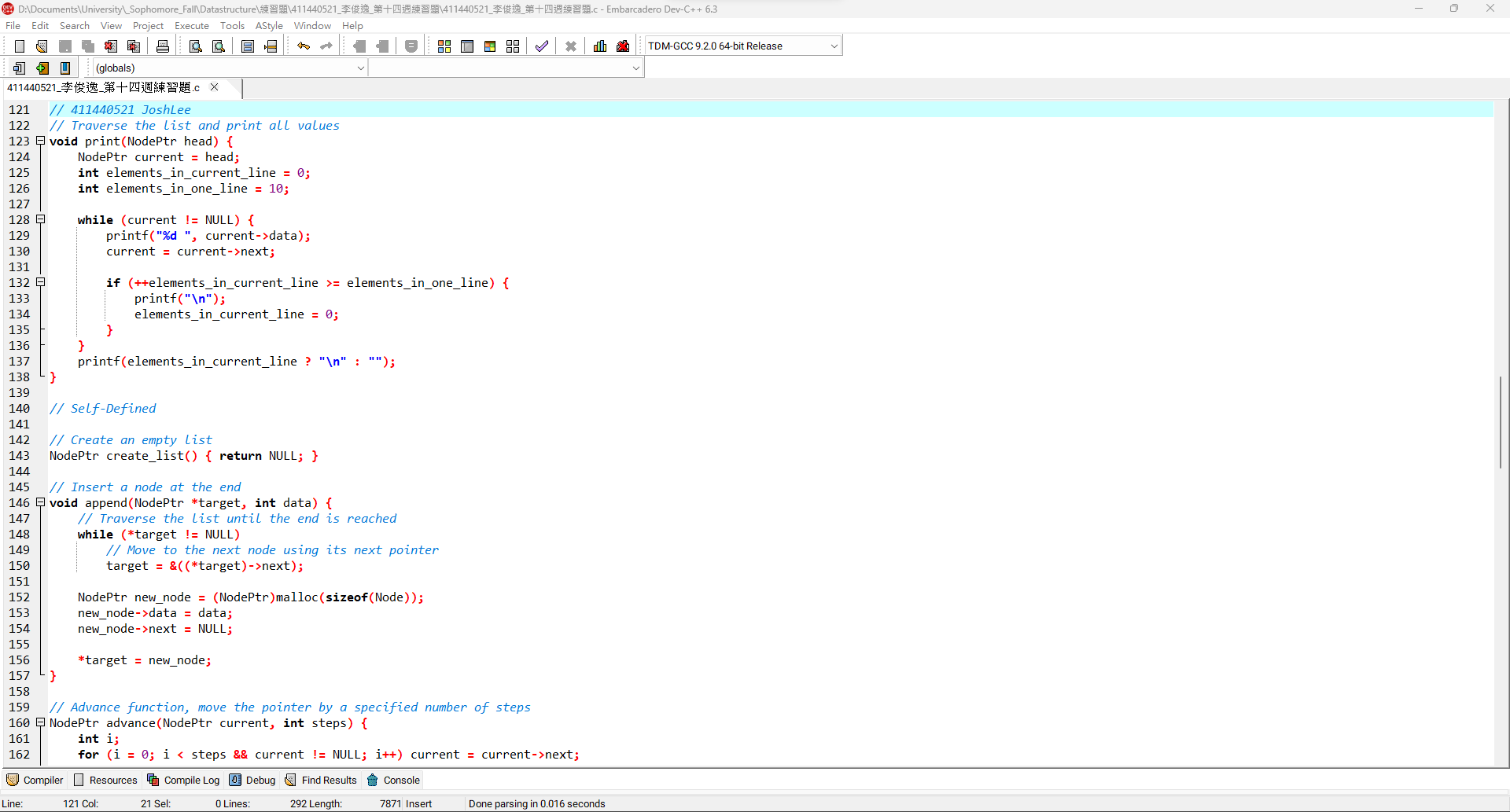
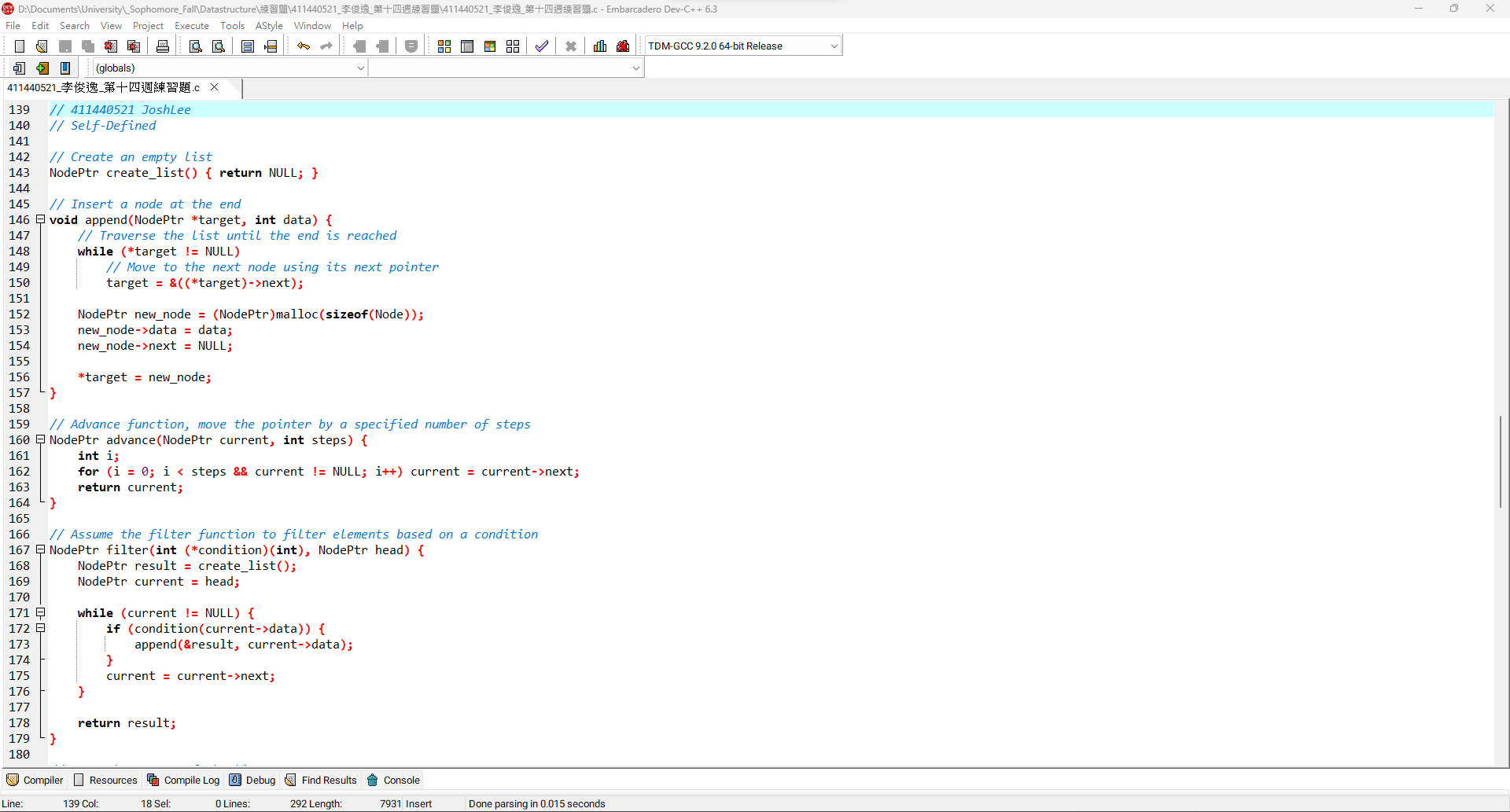
1

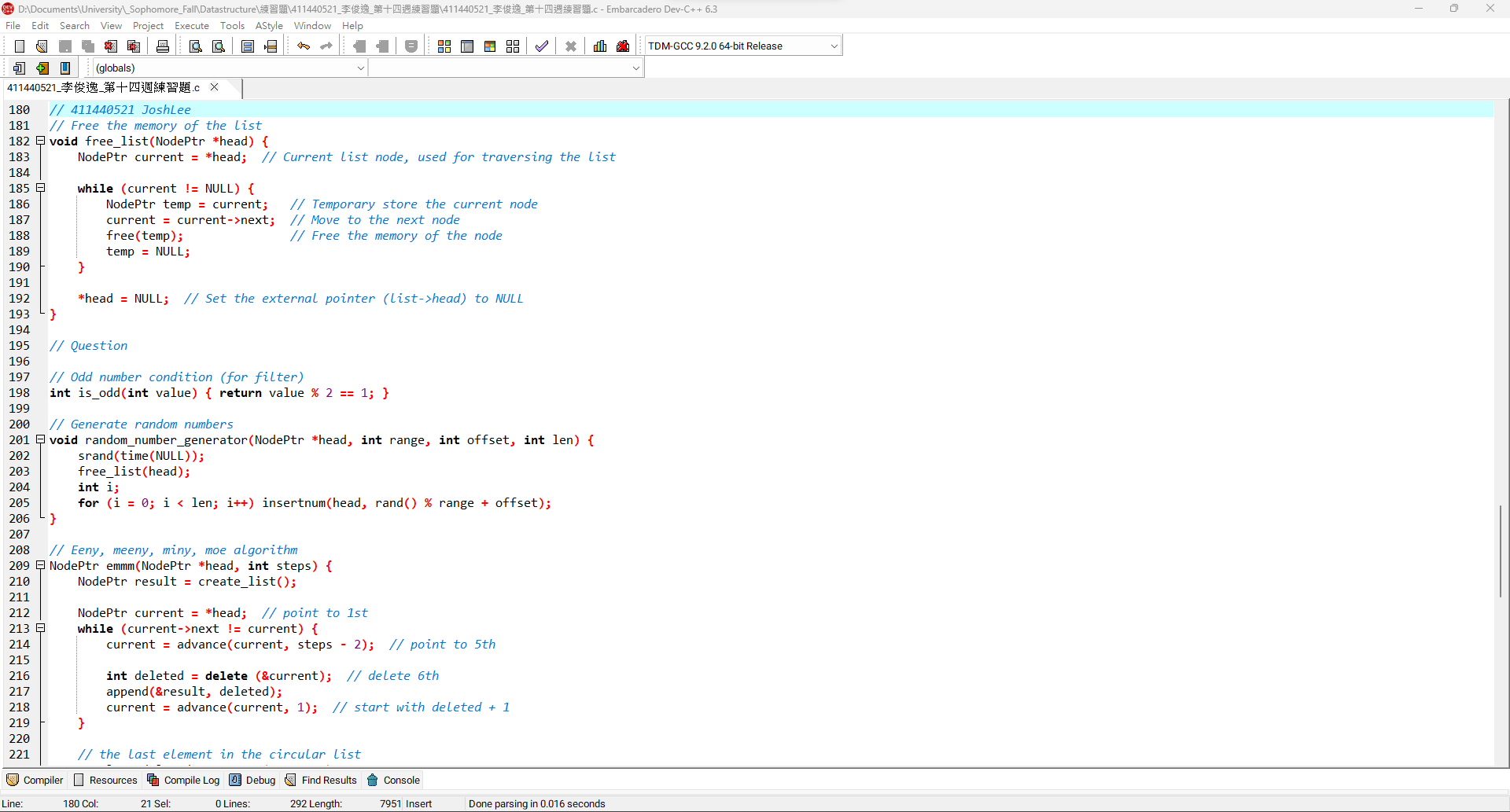
2

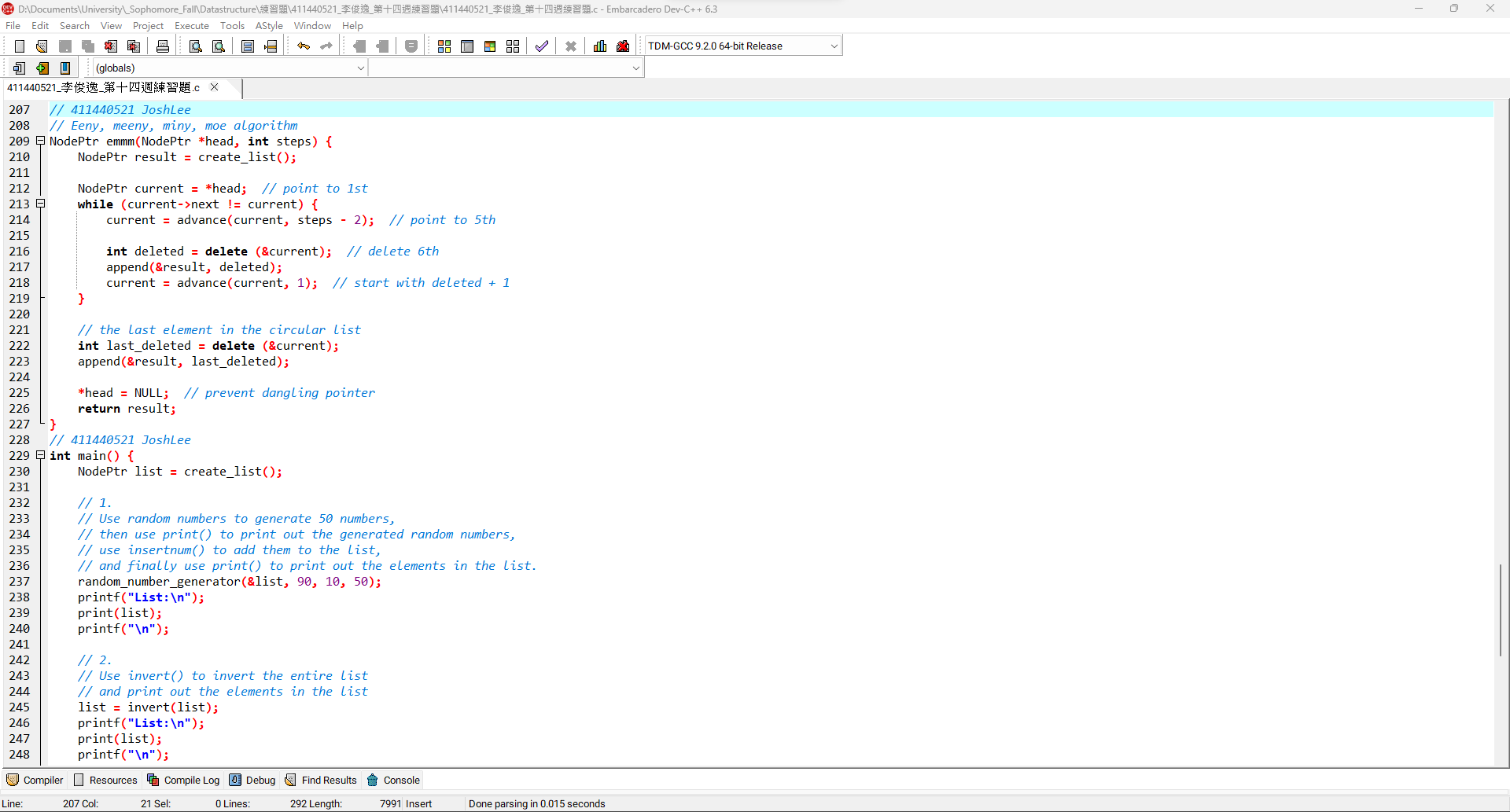
3

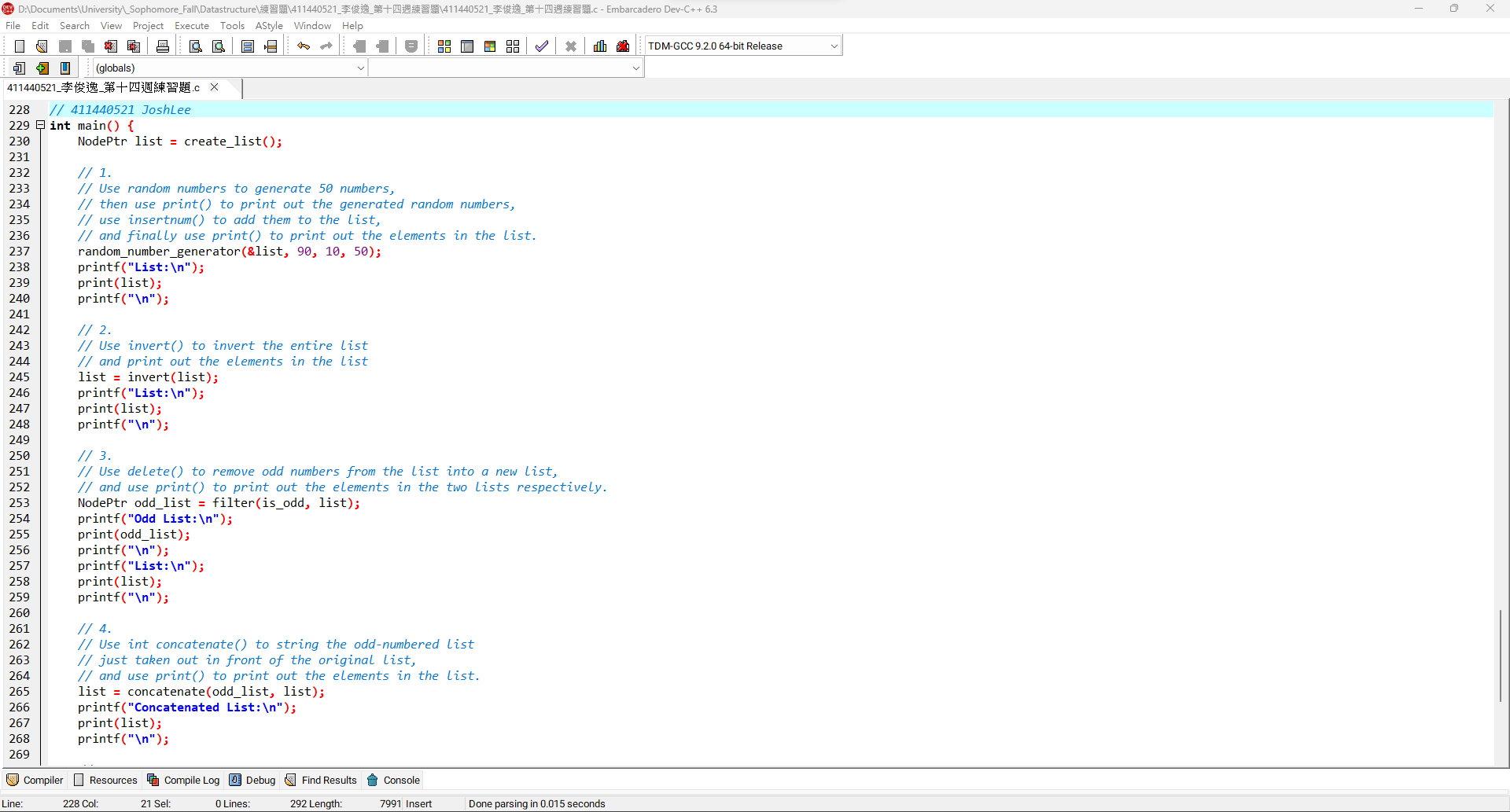
4

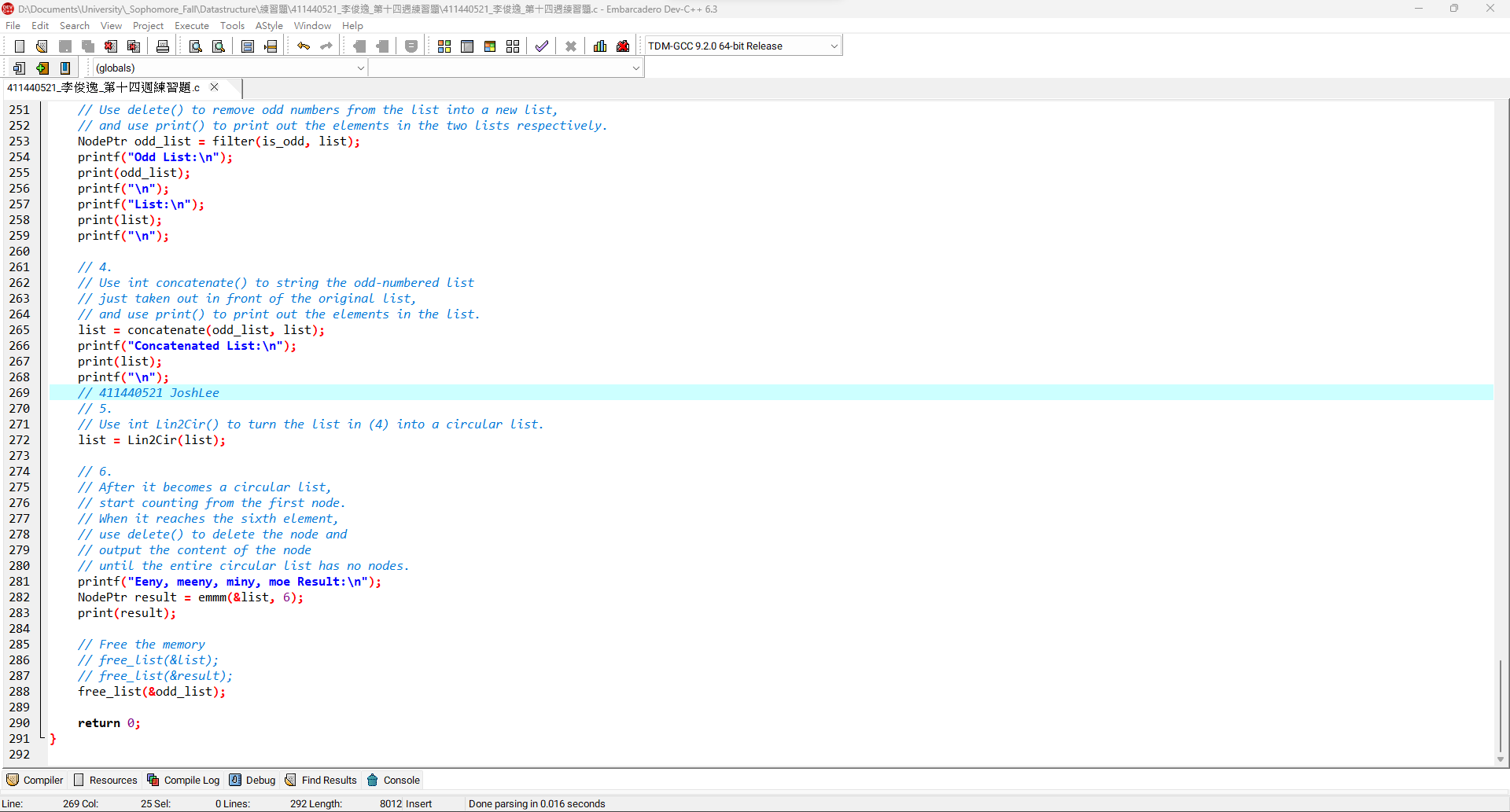
5

6

7

8

9

10

**程式碼:**

insert:

|  |
| --- |
| *// Insert a node behind*  void insert(NodePtr \*target, int data) {  NodePtr new\_node = (NodePtr)malloc(sizeof(Node));  new\_node->data = data;  new\_node->next = NULL;  *// Handle an empty list*  if (\*target == NULL) {  \*target = new\_node;  return;  }  *// Handle a non-empty list*  new\_node->next = (\*target)->next;  (\*target)->next = new\_node;  } |

delete

|  |
| --- |
| *// Delete the node behind and return its data*  int delete(NodePtr \*target) {  *// Handle an empty list*  if (\*target == NULL) {  printf("List is empty!");  exit(1);  }  *// Handle a list with only one element*  if ((\*target)->next == NULL) {  NodePtr deleted = \*target;  int data = deleted->data;  \*target = NULL;  free(deleted);  return data;  }  *// Handle a list with more than one element*  NodePtr deleted = (\*target)->next;  int data = deleted->data;  (\*target)->next = deleted->next;  free(deleted);  return data;  } |

insertnum:

|  |
| --- |
| *// Find the first position in the list not less than the given value*  NodePtr insertnum(NodePtr \*head, int value) {  NodePtr \*current = head; *// Use an indirect pointer*  *// Traverse the list to find the appropriate position*  while (\*current != NULL && (\*current)->data < value) {  current = &((\*current)->next);  }  *// Create a new node and insert it into the list*  NodePtr new\_node = (NodePtr)malloc(sizeof(Node));  new\_node->data = value;  new\_node->next = \*current;  \*current = new\_node;  return new\_node;  } |

concatenate:

|  |
| --- |
| *// Concatenate two lists*  NodePtr concatenate(NodePtr p, NodePtr q) {  NodePtr temp = p;  while (temp->next != NULL) {  temp = temp->next;  }  temp->next = q;  return p;  } |

invert:

|  |
| --- |
| *// Reverse the list*  NodePtr invert(NodePtr head) {  NodePtr prev = NULL;  NodePtr current = head;  NodePtr next\_node;  while (current != NULL) {  next\_node = current->next;  current->next = prev;  prev = current;  current = next\_node;  }  return prev;  } |

Lin2Cir:

|  |
| --- |
| *// Convert the list to a circular list*  NodePtr Lin2Cir(NodePtr head) {  NodePtr current = head;  *// Get the last node*  while (current->next != NULL) {  current = current->next;  }  *// Connect the last node to the head*  current->next = head;  return head;  } |

print:

|  |
| --- |
| *// Traverse the list and print all values*  void print(NodePtr head) {  NodePtr current = head;  int elements\_in\_current\_line = 0;  int elements\_in\_one\_line = 10;  while (current != NULL) {  printf("%d ", current->data);  current = current->next;  if (++elements\_in\_current\_line >= elements\_in\_one\_line) {  printf("\n");  elements\_in\_current\_line = 0;  }  }  printf(elements\_in\_current\_line ? "\n" : "");  } |