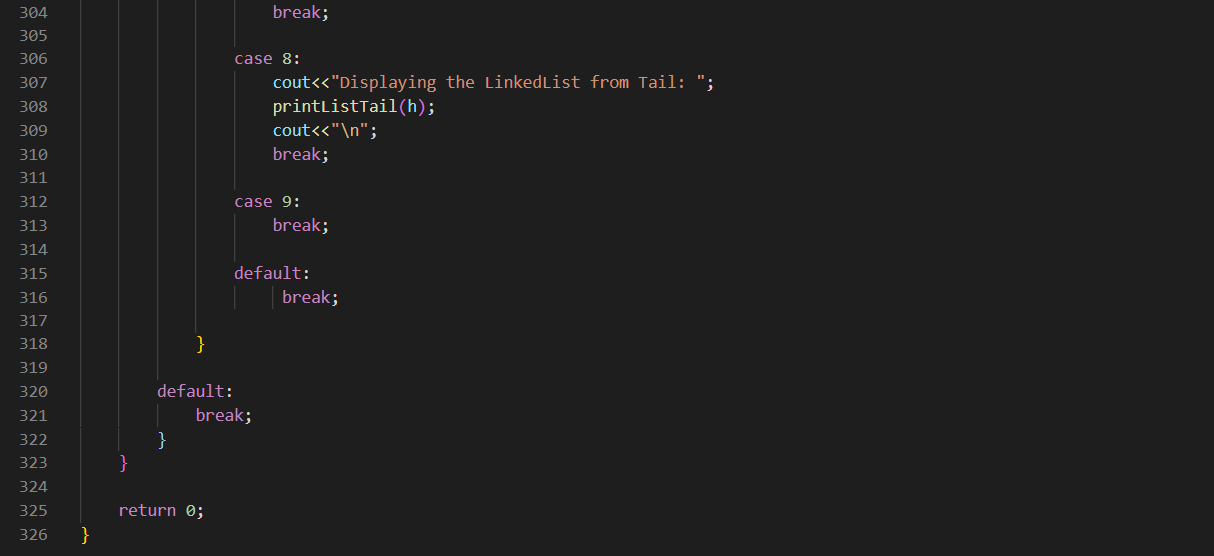
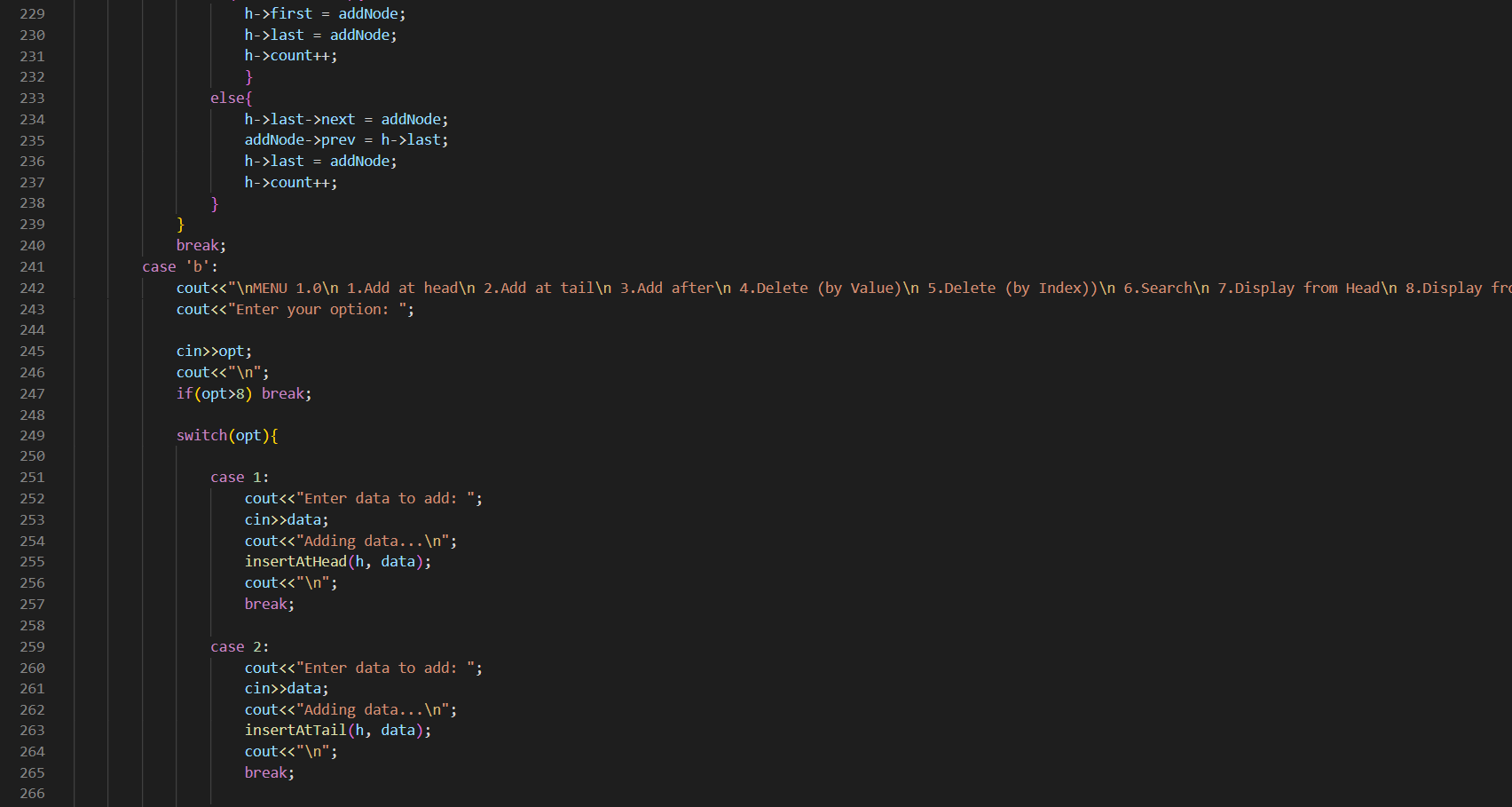
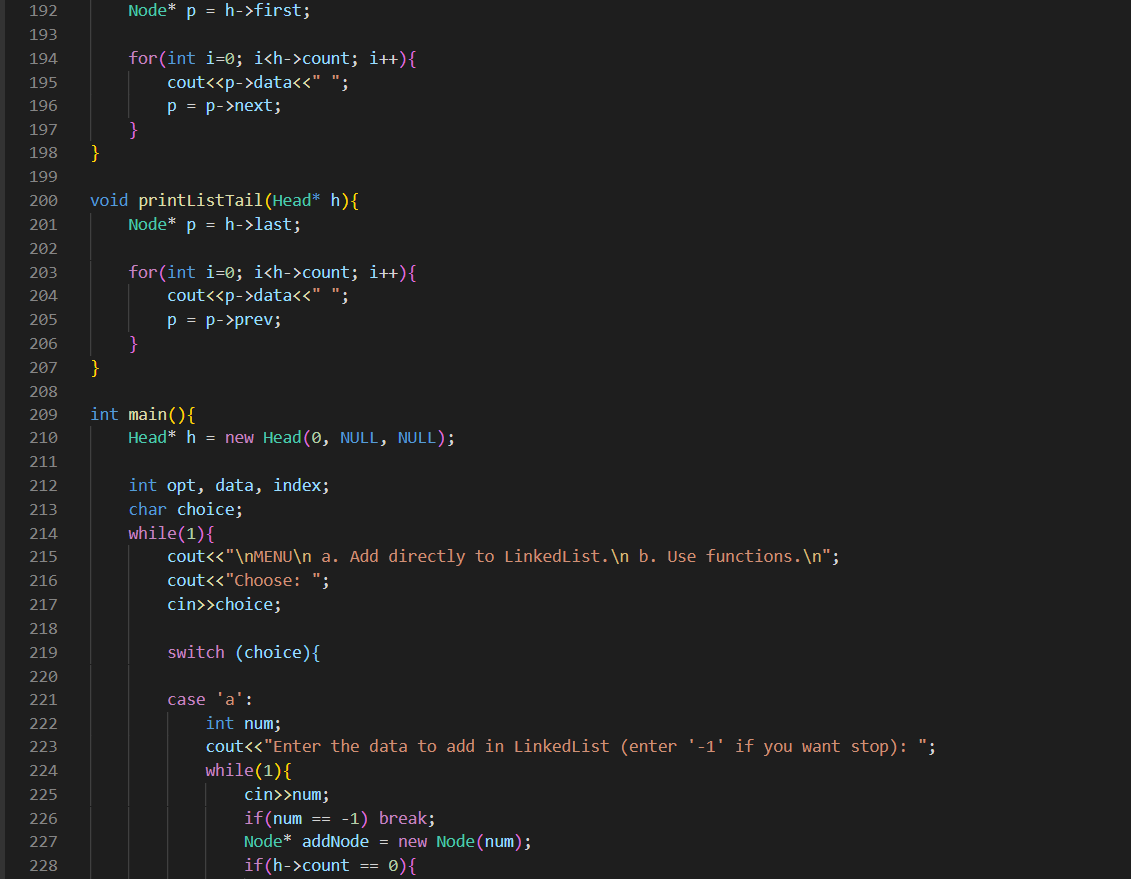
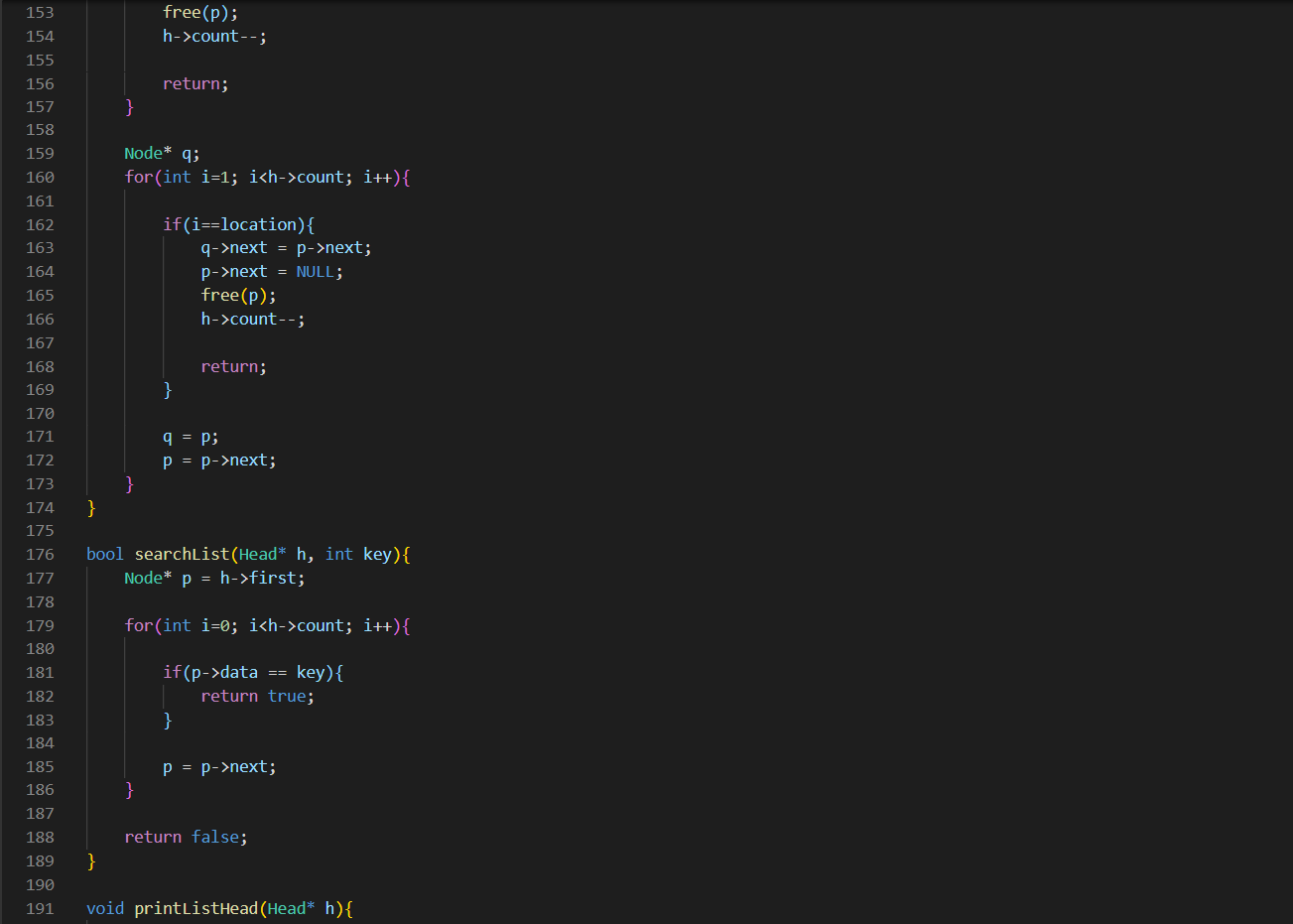
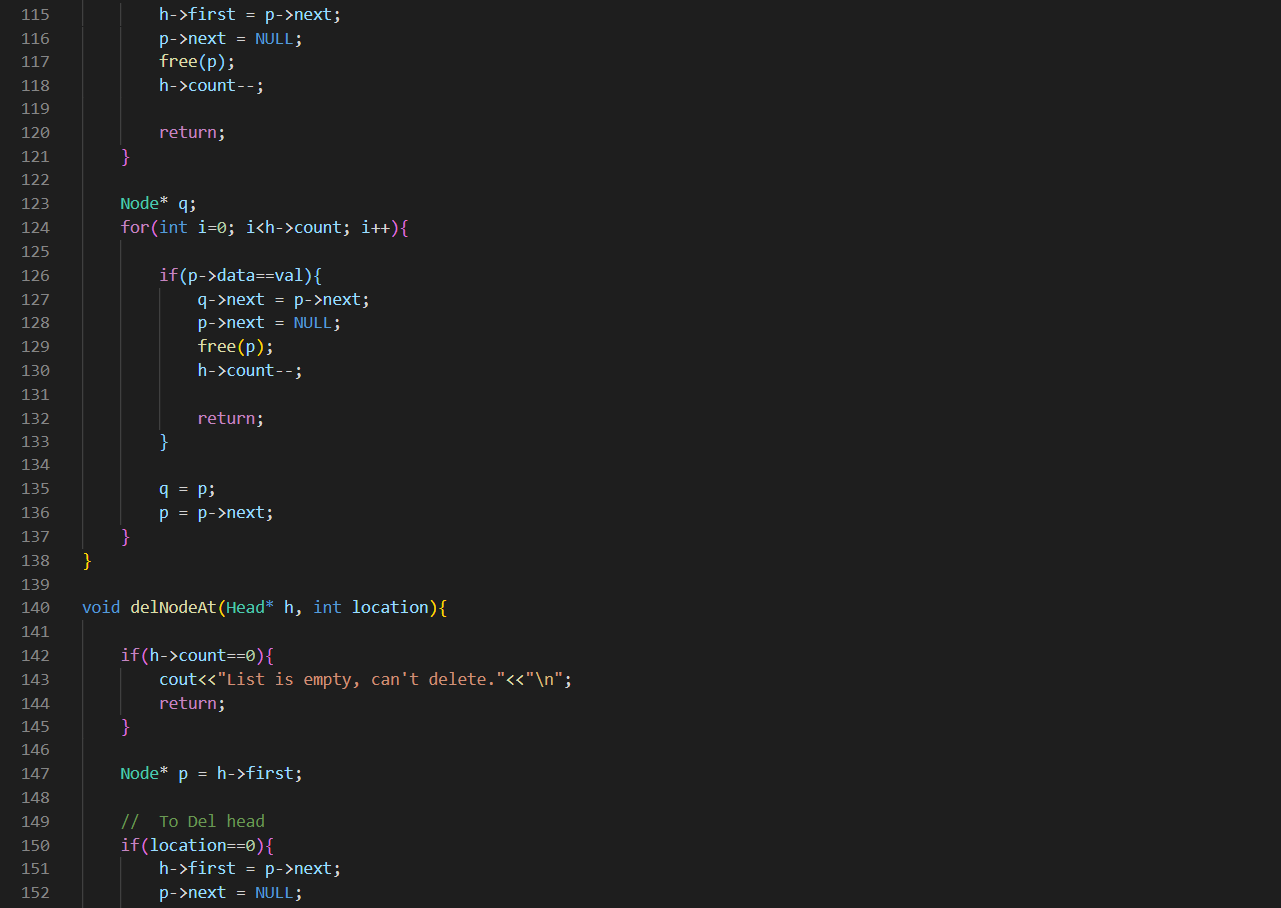
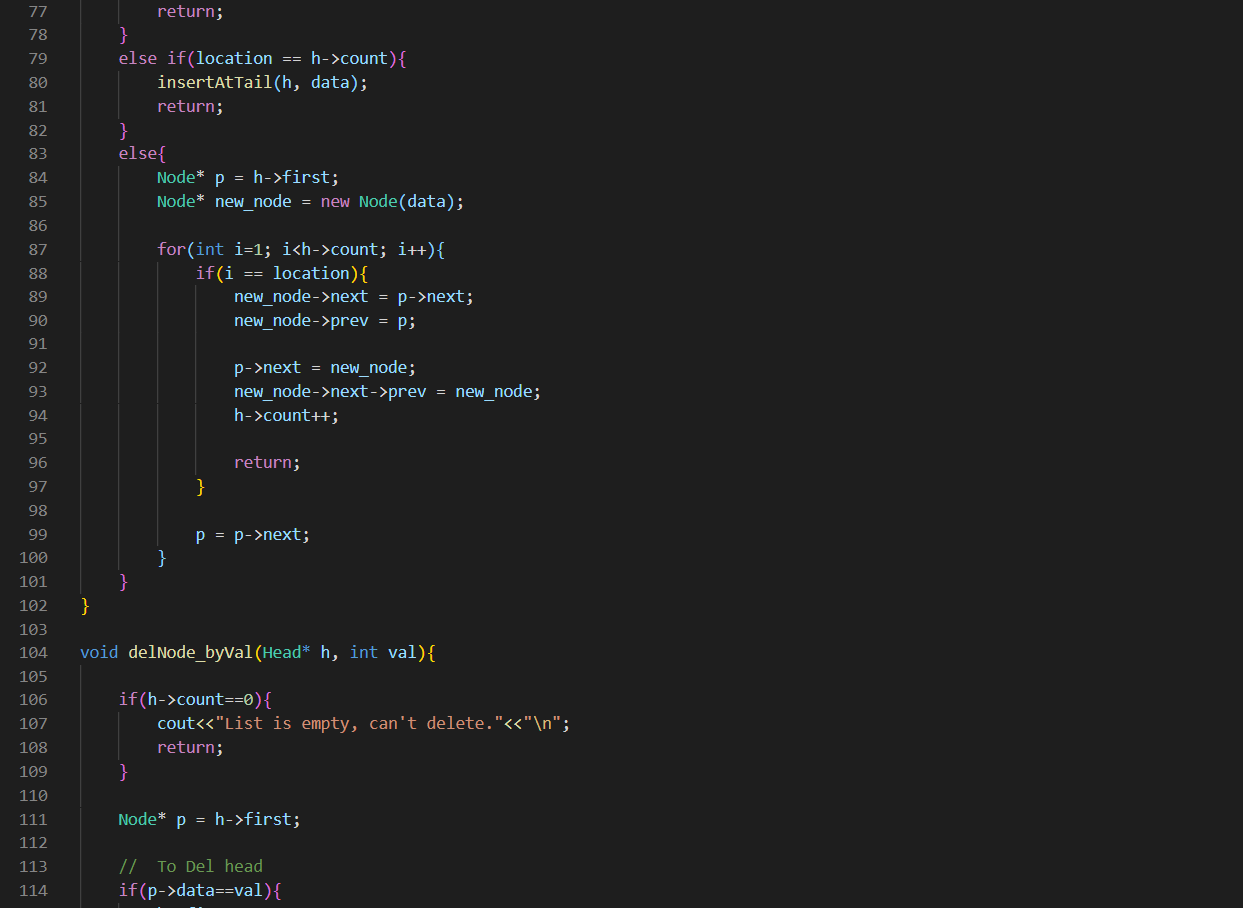
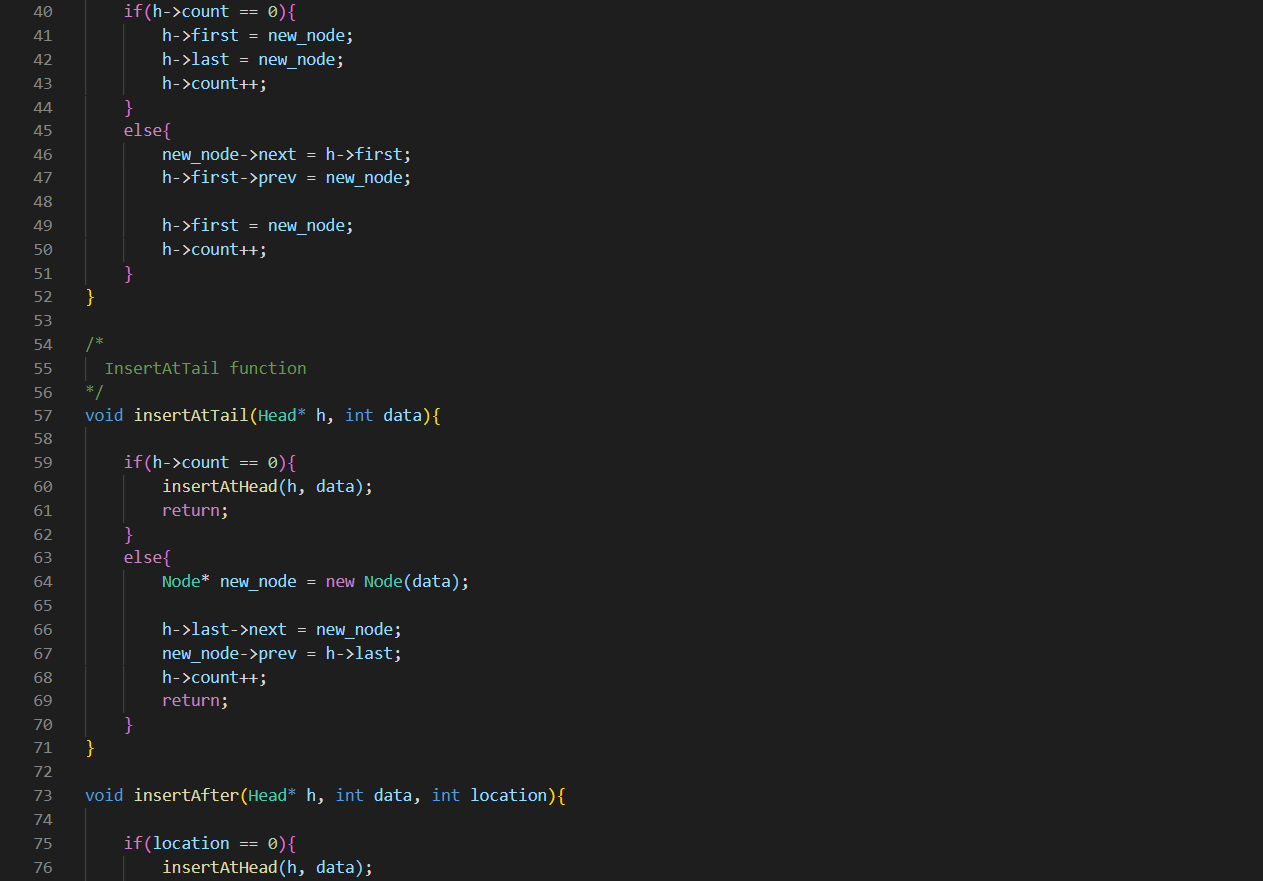
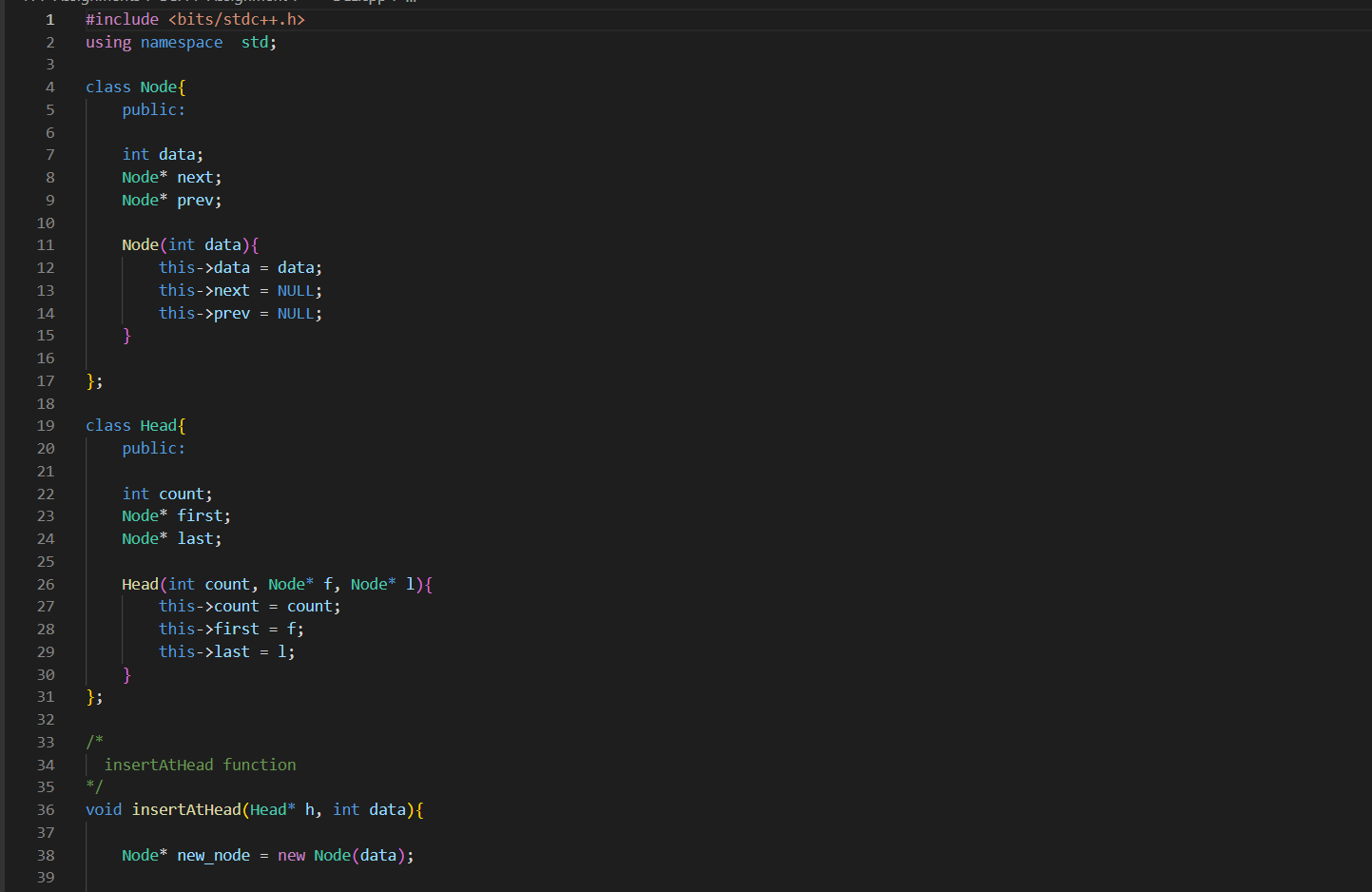
PRN: 2020BTEIT00041

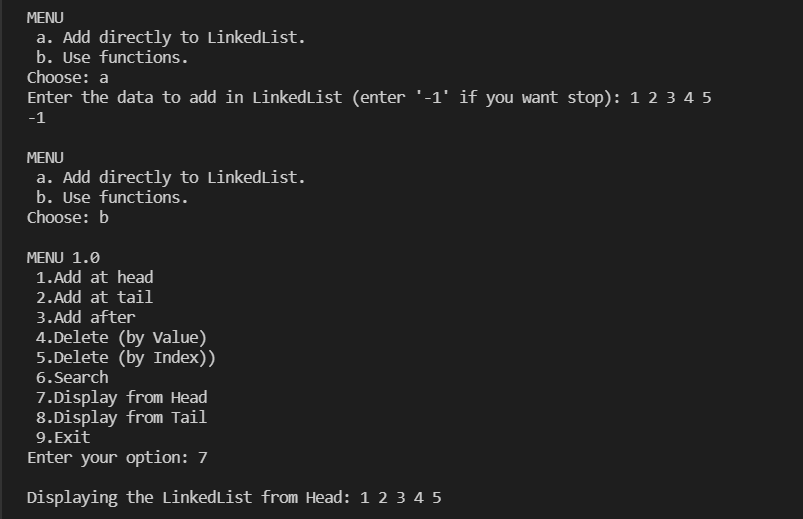
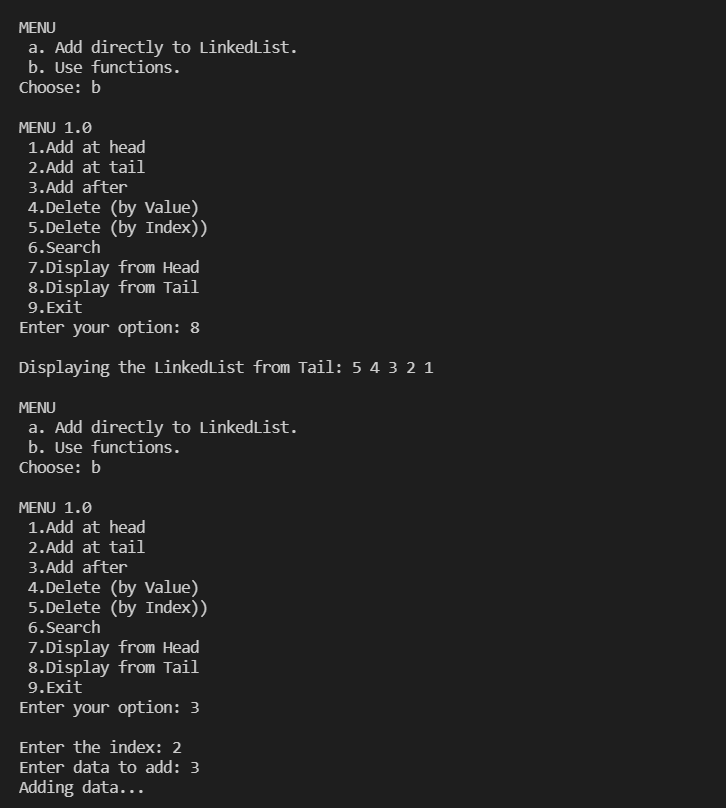
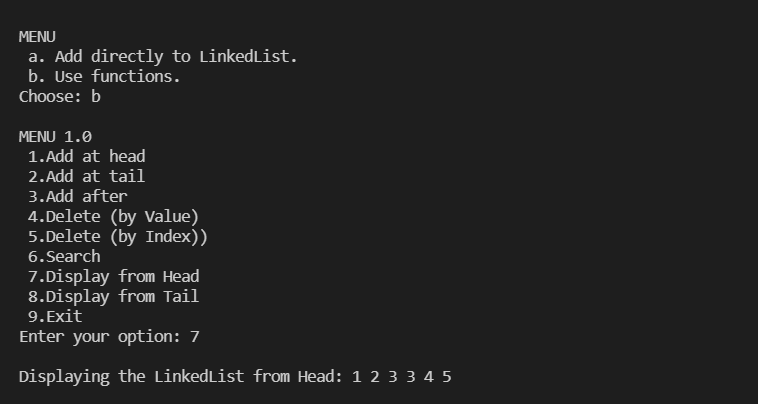
NAME: Om Vivek Gharge

Q.) Doubly Linked List:

CODE:



OUTPUT:

ALGORITHM:

Algorithm to create Doubly Linked list

Begin:

alloc (head)

If (head == NULL) then

write ('Unable to allocate memory')

End if

Else then

read (data)

head.data ← data;

head.prev ← NULL;

head.next ← NULL;

last ← head;

write ('List created successfully')

End else

End

Algorithm to traverse Doubly Linked list

Input : head {Pointer to the first node of the list}

Begin:

If (head == NULL) then

write ('List is empty')

End if

Else then

temp ← head;

While (temp != NULL) do

write ('Data = ', temp.data)

temp ← temp.next;

End while

End else

End

Algorithm to insert a node at the beginning:

Input : head {A pointer pointing to the first node of the list}

Begin:

alloc (newNode)

If (newNode == NULL) then

write ('Unable to allocate memory')

End if

Else then

read (data)

newNode.data ← data;

newNode.prev ← NULL;

newNode.next ← head;

head.prev ← newNode;

head ← newNode;

write('Node added successfully at the beginning of List')

End else

End

Algorithm to insert a node at the end of Doubly linked list

Input : last {Pointer to the last node of doubly linked list}

Begin:

alloc (newNode)

If (newNode == NULL) then

write ('Unable to allocate memory')

End if

Else then

read (data)

newNode.data ← data;

newNode.next ← NULL;

newNode.prev ← last;

last.next ← newNode;

last ← newNode;

write ('Node added successfully at the end of List')

End else

End

Algorithm to insert node at any position of doubly linked list

Input : head {Pointer to the first node of doubly linked list}

: last {Pointer to the last node of doubly linked list}

: N {Position where node is to be inserted}

Begin:

temp ← head

For i←1 to N-1 do

If (temp == NULL) then

break

End if

temp ← temp.next;

End for

If (N == 1) then

insertAtBeginning()

End if

Else If (temp == last) then

insertAtEnd()

End if

Else If (temp != NULL) then

alloc (newNode)

read (data)

newNode.data ← data;

newNode.next ← temp.next

newNode.prev ← temp

If (temp.next != NULL) then

temp.next.prev ← newNode;

End if

temp.next ← newNode;

write('Node added successfully')

End if

End

Algorithm to delete node from beginning

Input: head {Pointer to first node of the linked list}

Begin:

If (head == NULL) then

write ('Can't delete from an empty list')

End if

Else then

toDelete ← head;

head ← head.next;

head.prev ← NULL;

unalloc (toDelete)

write ('Successfully deleted first node from the list')

End if

End

Algorithm to delete node from end

Input: last {Pointer to last node of the linked list}

Begin:

If (last == NULL) then

write ('Can't delete from an empty list')

End if

Else then

toDelete ← last;

last ← last.prev;

last.next ← NULL;

unalloc (toDelete)

write ('Successfully deleted last node from the list')

End if

End

Algorithm to delete node from any position

Input : head {Pointer to the first node of the list}

last {Pointer to the last node of the list}

N {Position to be deleted from list}

Begin:

current ← head;

For i ← 1 to N and current != NULL do

current ← current.next;

End for

If (N == 1) then

deleteFromBeginning()

End if

Else if (current == last) then

deleteFromEnd()

End if

Else if (current != NULL) then

current.prev.next ← current.next

If (current.next != NULL) then

current.next.prev ← current.prev;

End if

unalloc (current)

write ('Node deleted successfully from ', N, ' position')

End if

Else then

write ('Invalid position')

End if

End

Algorithm to insert a node at the beginning:

Input : head {A pointer pointing to the first node of the list}

Data {Data to search}

Begin:

alloc (newNode)

while(newNode != NULL)

{

if(newNode->data == Data) then

return 1;

newNode = newNode->next;

}

return 0;

End