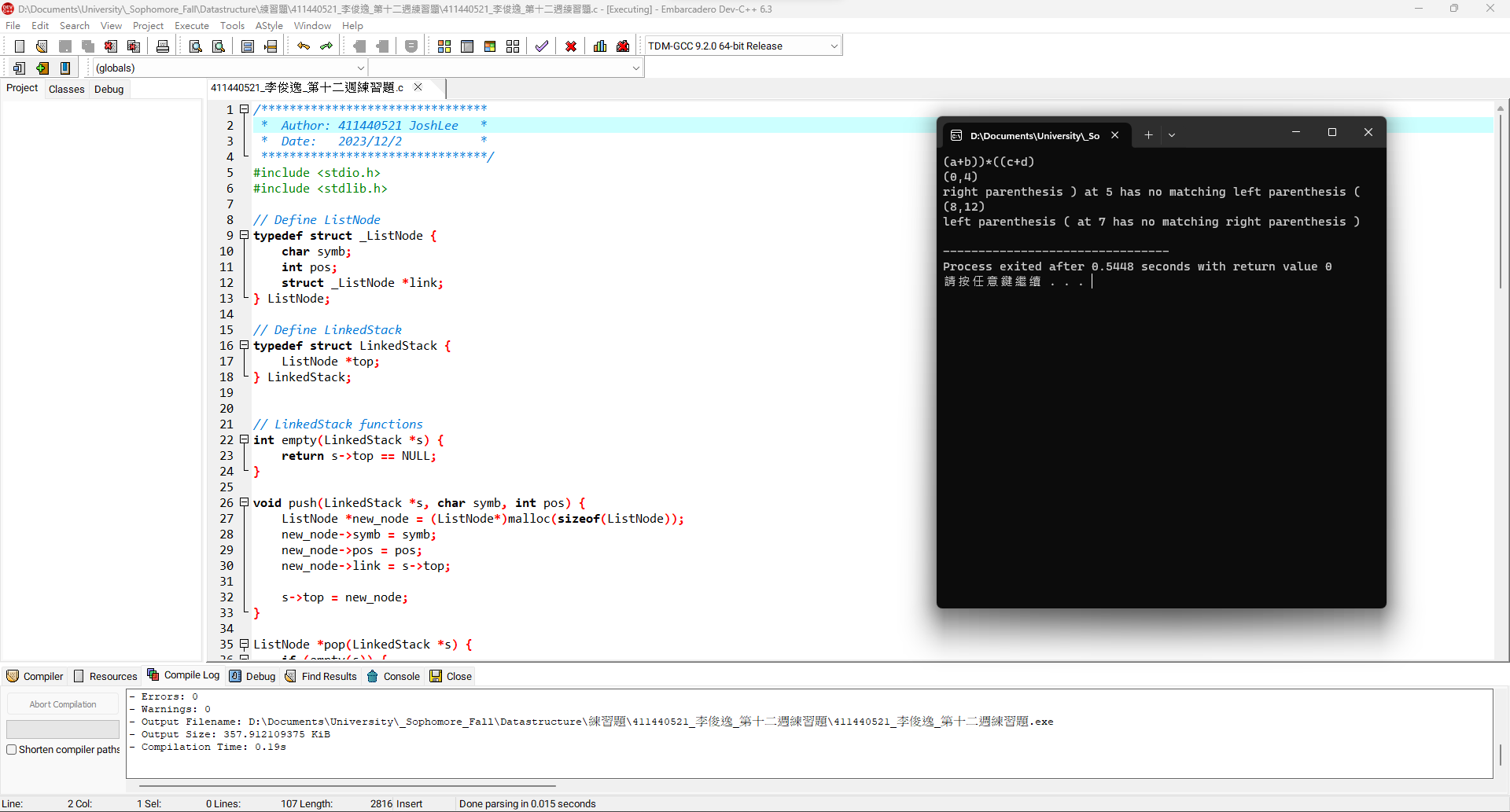
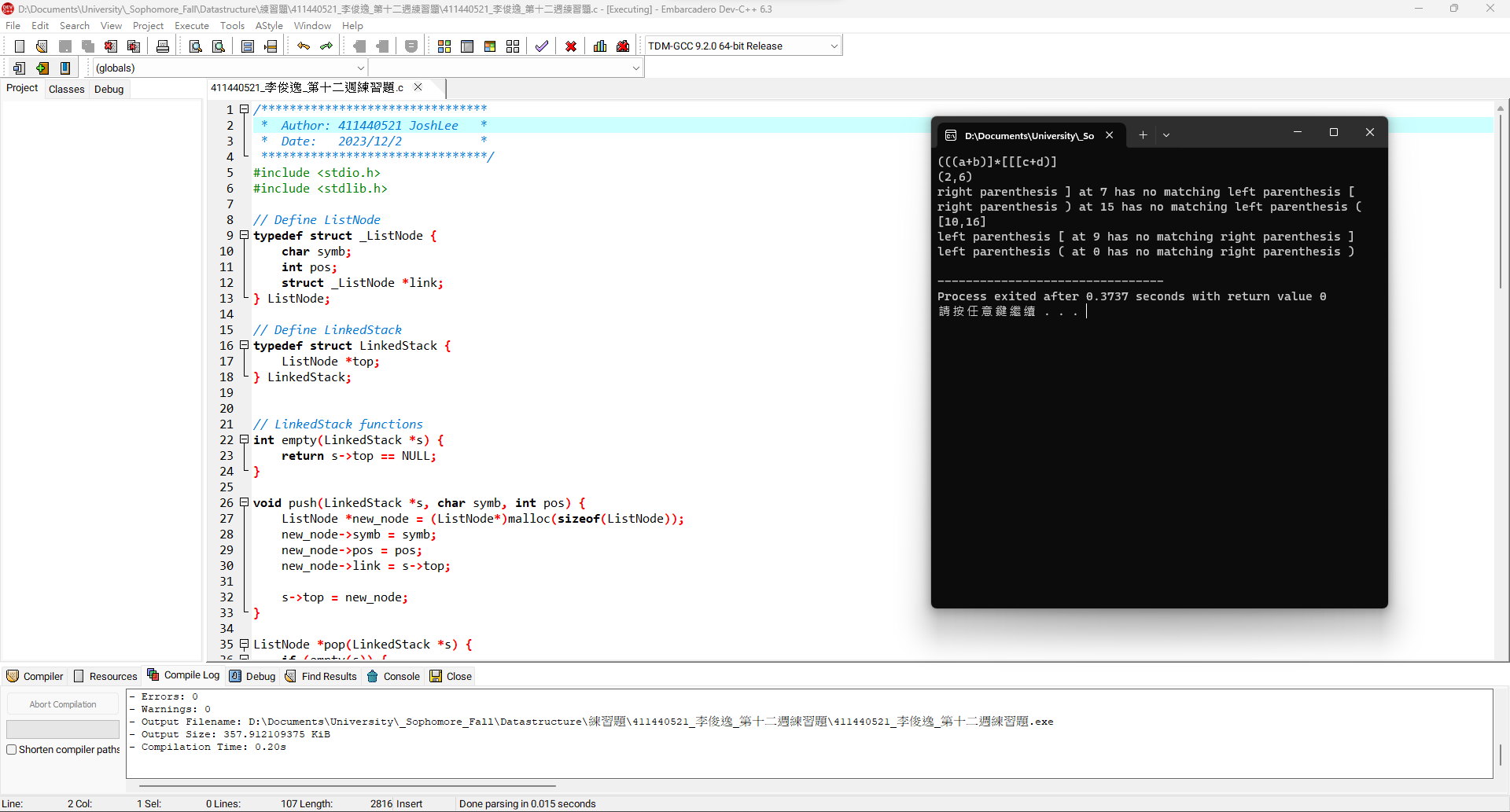
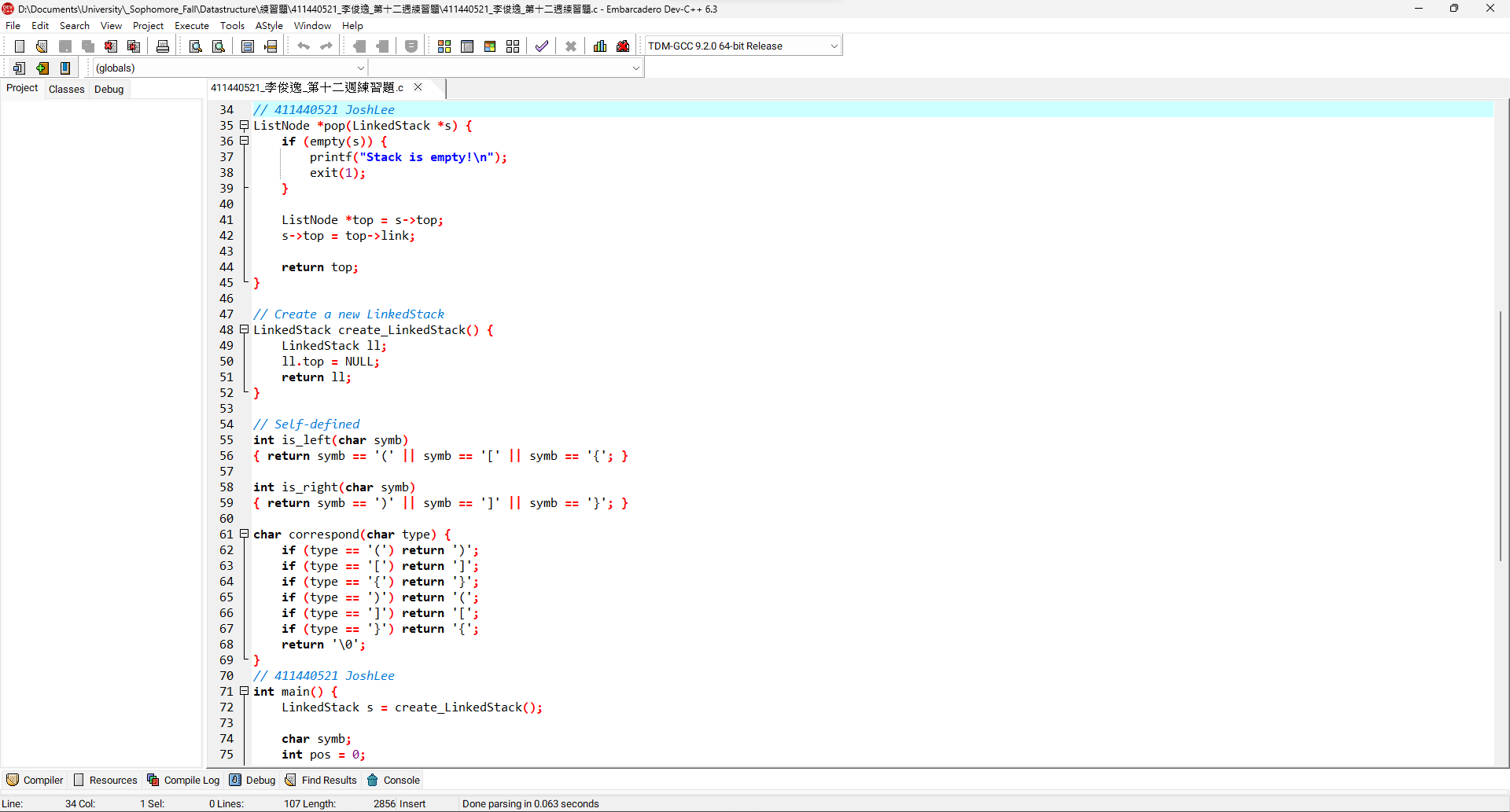
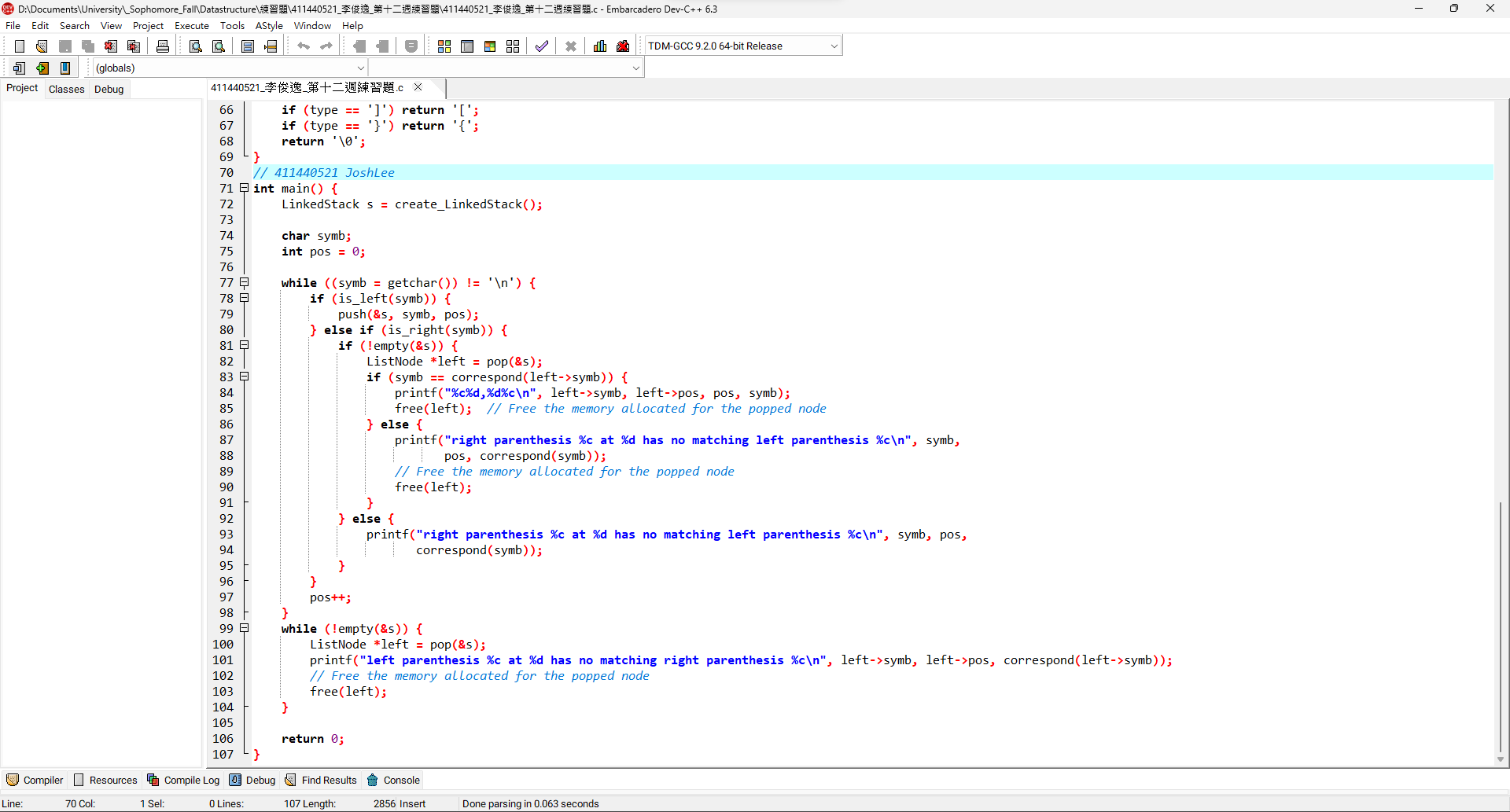
1

2

3

4

5

程式碼:

|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Author: 411440521 JoshLee \**  *\* Date: 2023/12/2 \**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  #include <stdio.h>  #include <stdlib.h>  *// Define ListNode*  typedef struct \_ListNode {  char symb;  int pos;  struct \_ListNode \*link;  } ListNode;  *// Define LinkedStack*  typedef struct LinkedStack {  ListNode \*top;  } LinkedStack;  *// LinkedStack functions*  int empty(LinkedStack \*s) {  return s->top == NULL;  }  void push(LinkedStack \*s, char symb, int pos) {  ListNode \*new\_node = (ListNode\*)malloc(sizeof(ListNode));  new\_node->symb = symb;  new\_node->pos = pos;  new\_node->link = s->top;    s->top = new\_node;  }  ListNode \*pop(LinkedStack \*s) {  if (empty(s)) {  printf("Stack is empty!\n");  exit(1);  }  ListNode \*top = s->top;  s->top = top->link;    return top;  }  *// Create a new LinkedStack*  LinkedStack create\_LinkedStack() {  LinkedStack ll;  ll.top = NULL;  return ll;  }  *// Self-defined*  int is\_left(char symb)  { return symb == '(' || symb == '[' || symb == '{'; }  int is\_right(char symb)  { return symb == ')' || symb == ']' || symb == '}'; }  char correspond(char type) {  if (type == '(') return ')';  if (type == '[') return ']';  if (type == '{') return '}';  if (type == ')') return '(';  if (type == ']') return '[';  if (type == '}') return '{';  return '\0';  }  int main() {  LinkedStack s = create\_LinkedStack();  char symb;  int pos = 0;  while ((symb = getchar()) != '\n') {  if (is\_left(symb)) {  push(&s, symb, pos);  } else if (is\_right(symb)) {  if (!empty(&s)) {  ListNode \*left = pop(&s);  if (symb == correspond(left->symb)) {  printf("%c%d,%d%c\n", left->symb, left->pos, pos, symb);  free(left); *// Free the memory allocated for the popped node*  } else {  printf("right parenthesis %c at %d has no matching left parenthesis %c\n", symb,  pos, correspond(symb));  *// Free the memory allocated for the popped node*  free(left);  }  } else {  printf("right parenthesis %c at %d has no matching left parenthesis %c\n", symb, pos,  correspond(symb));  }  }  pos++;  }  while (!empty(&s)) {  ListNode \*left = pop(&s);  printf("left parenthesis %c at %d has no matching right parenthesis %c\n", left->symb, left->pos, correspond(left->symb));  *// Free the memory allocated for the popped node*  free(left);  }  return 0;  } |