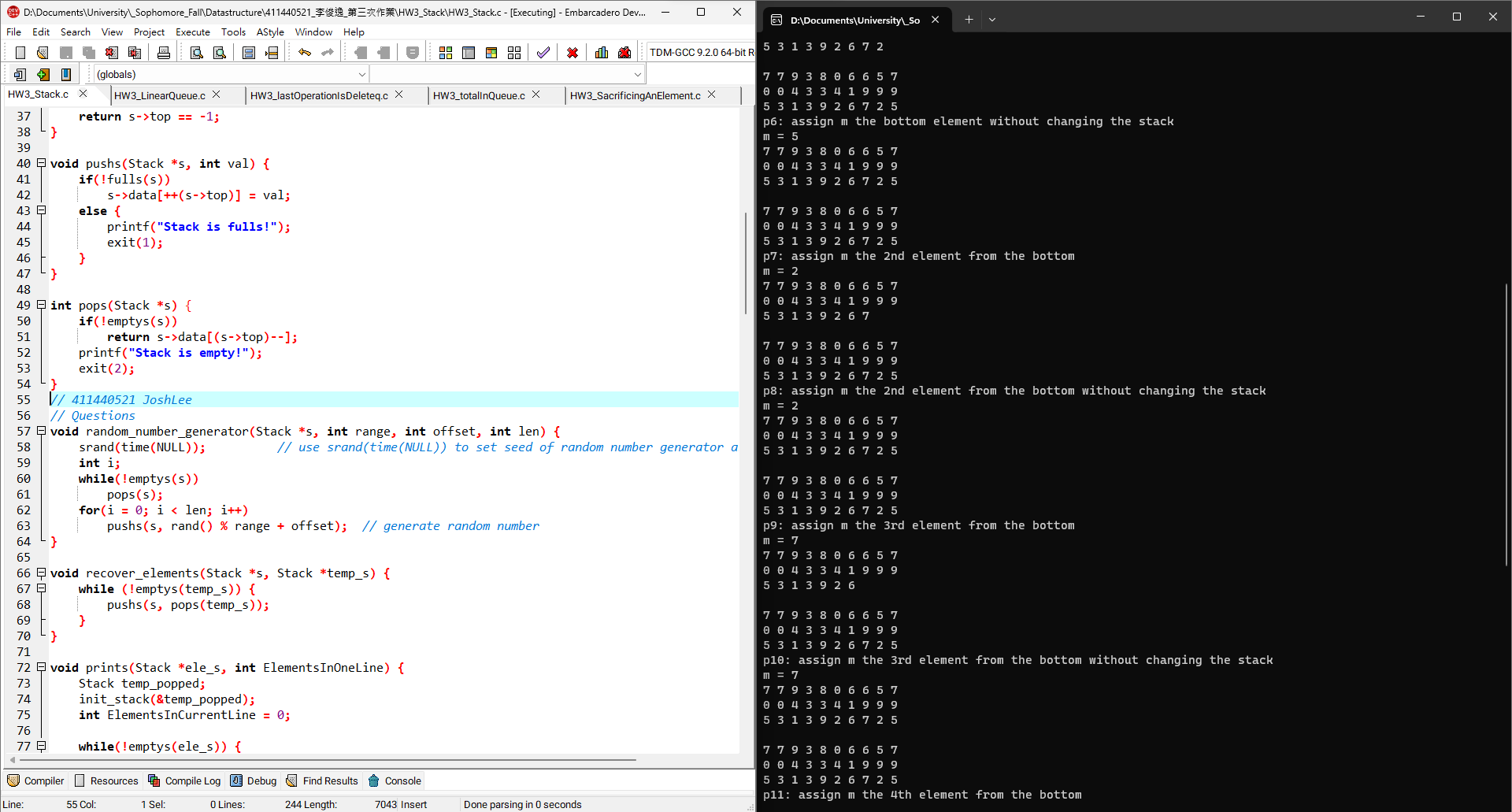
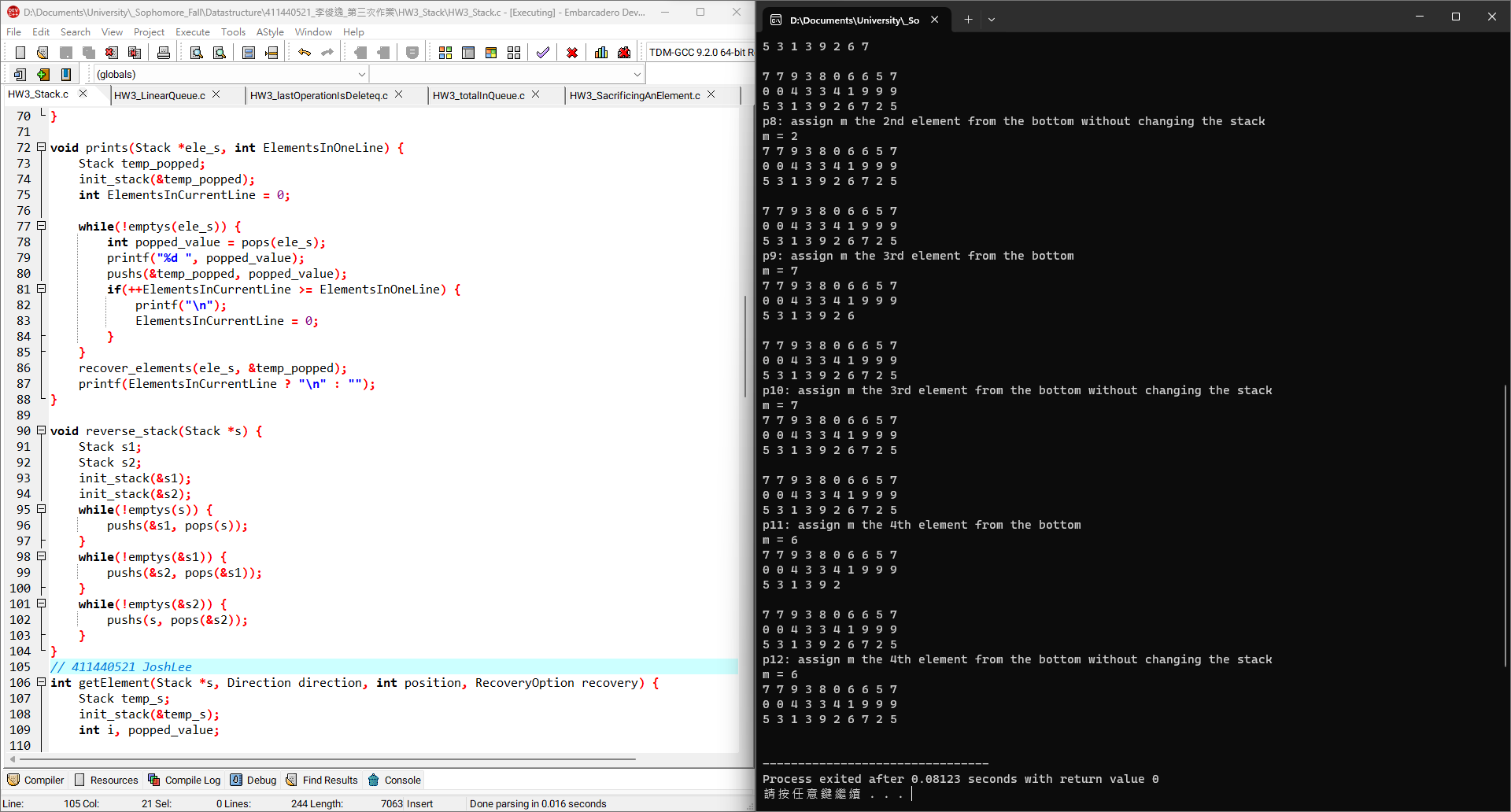
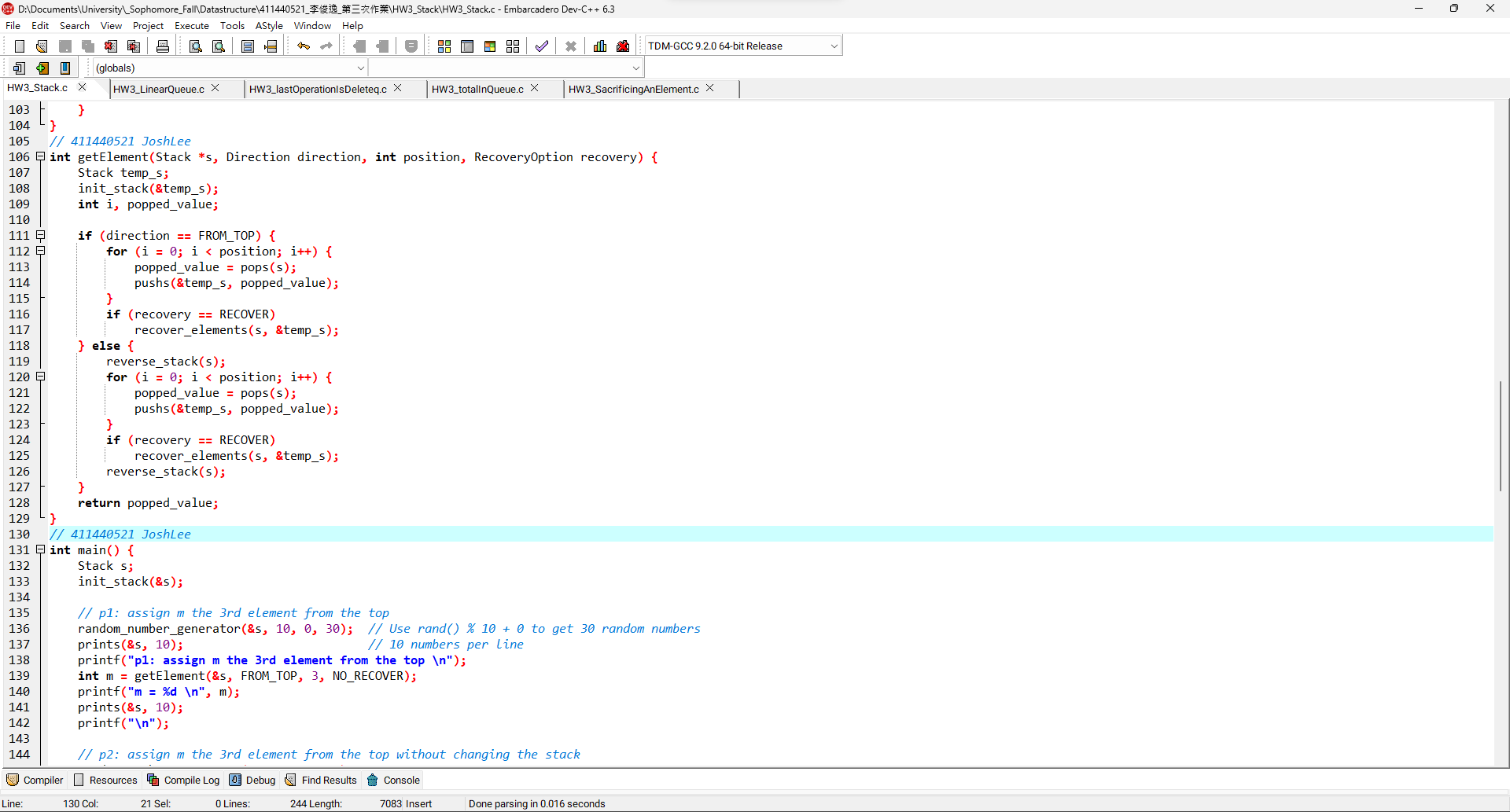
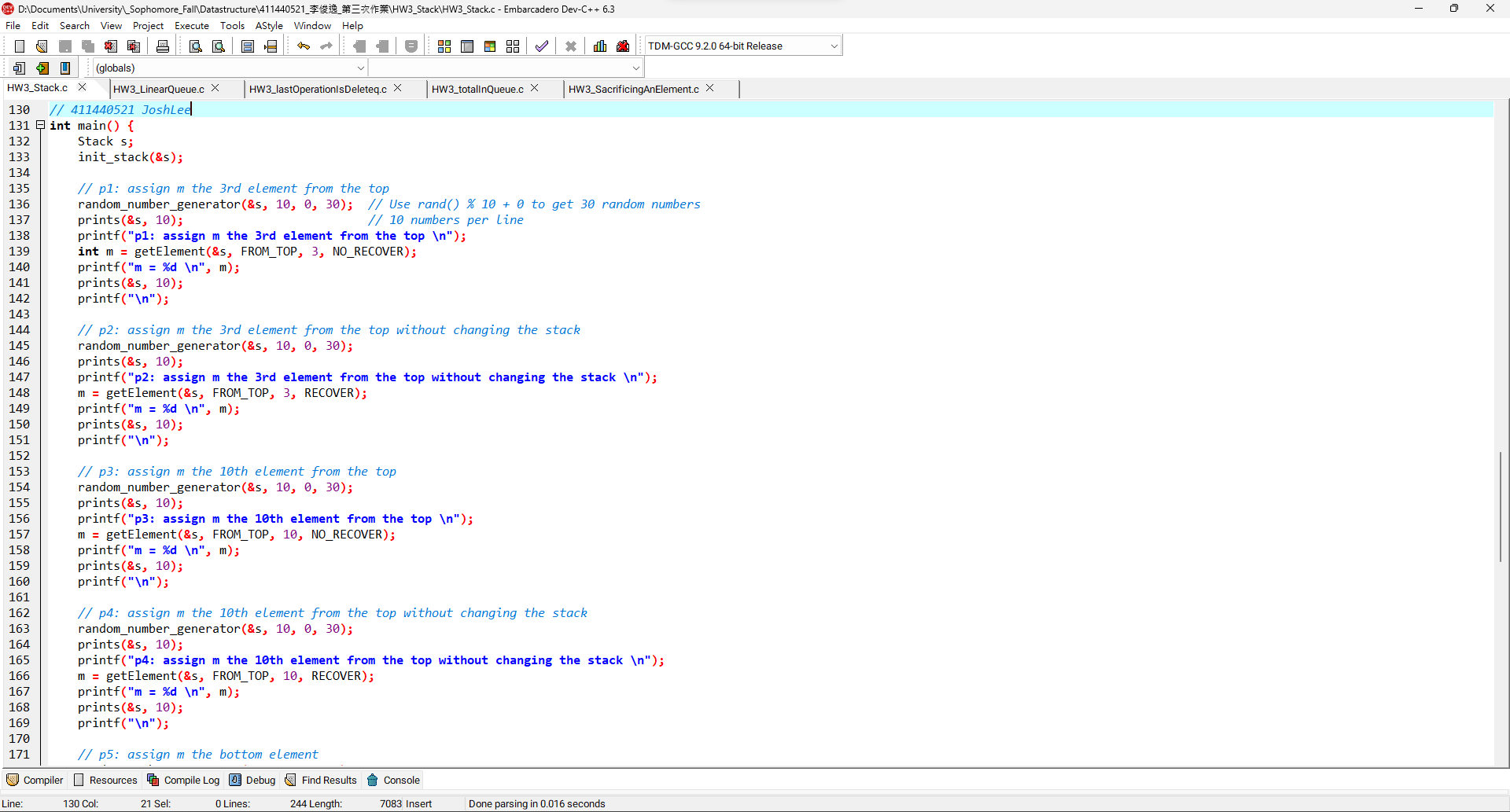
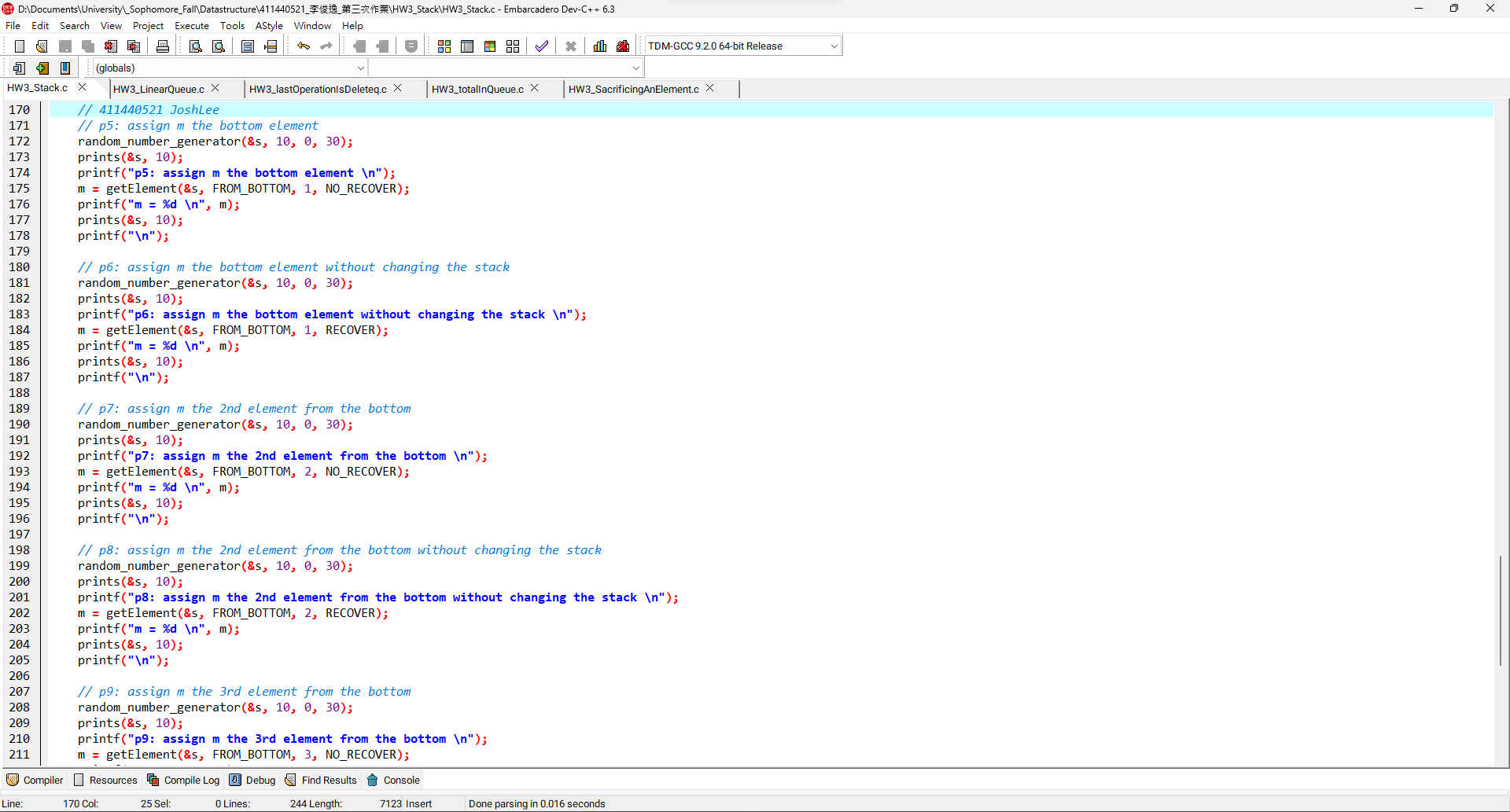
1

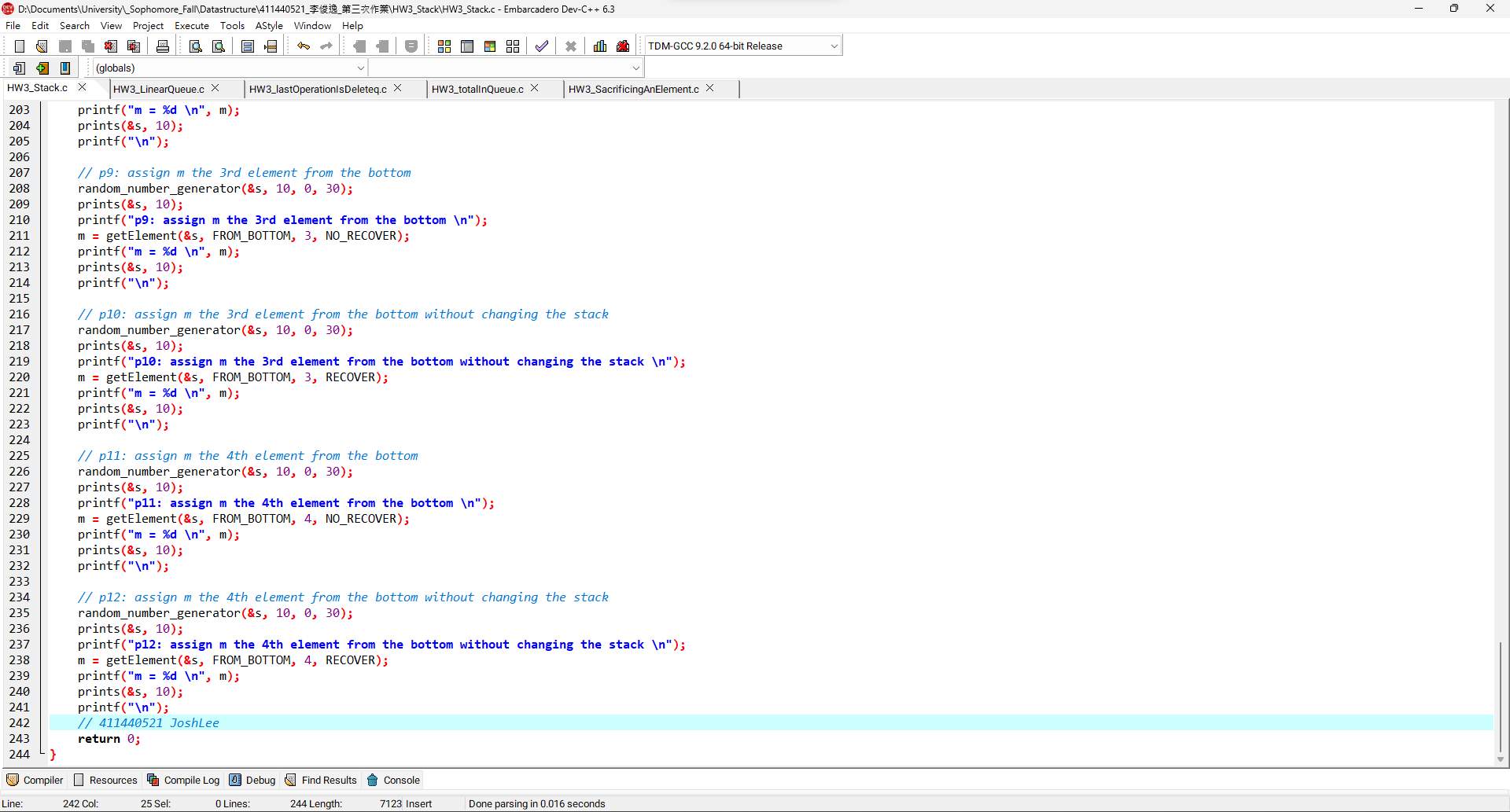
2

3

4

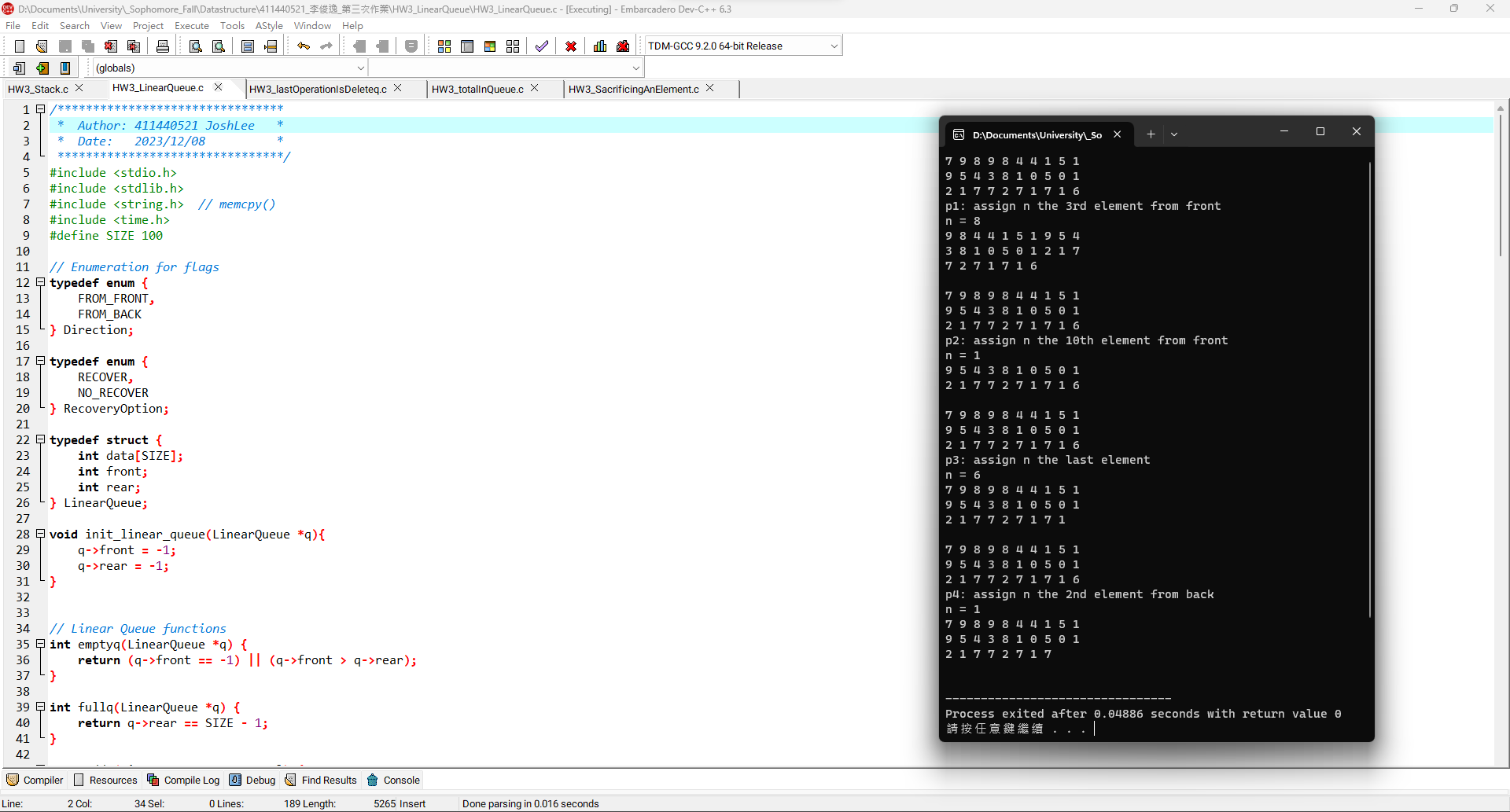
5

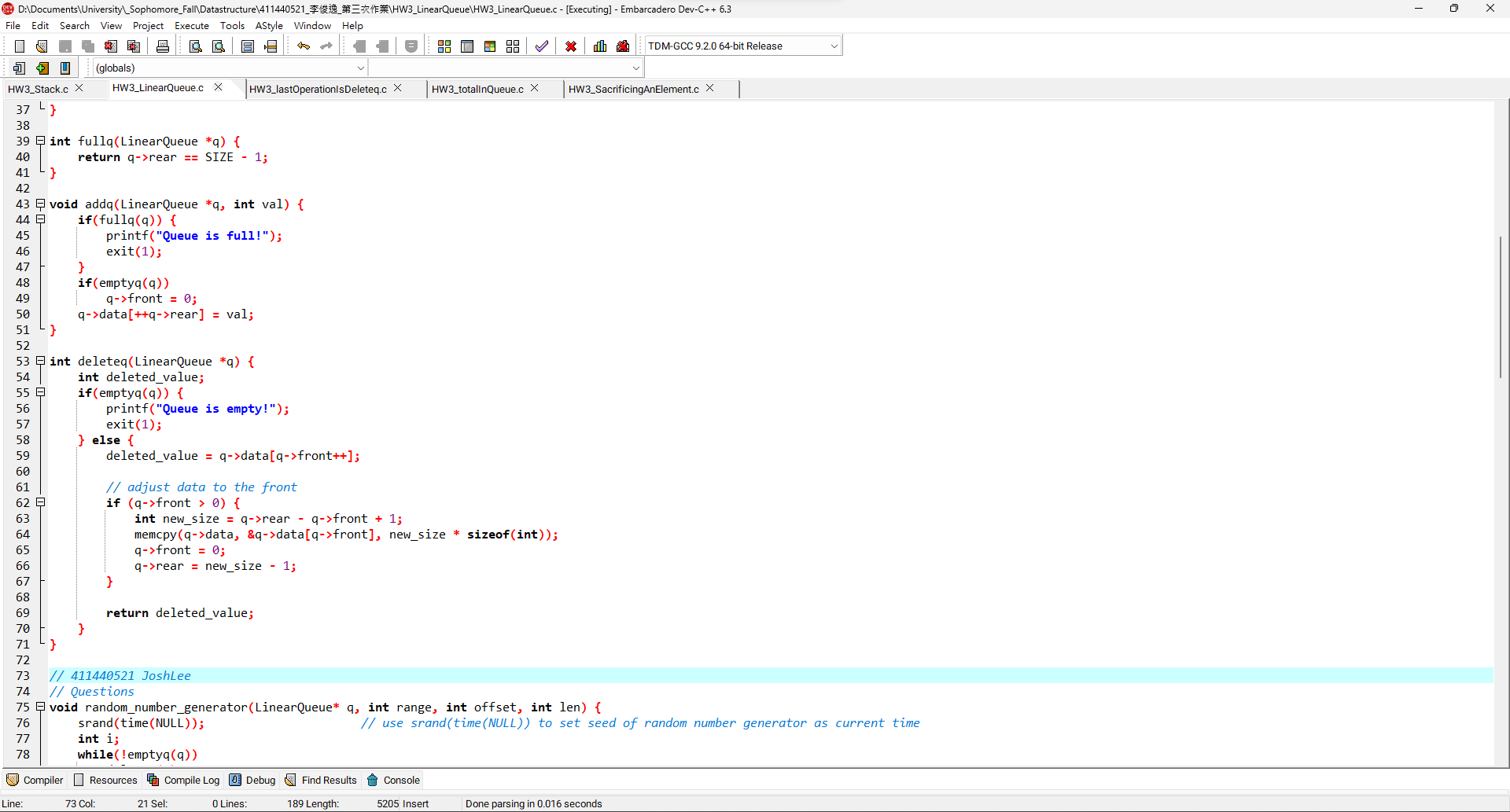
6

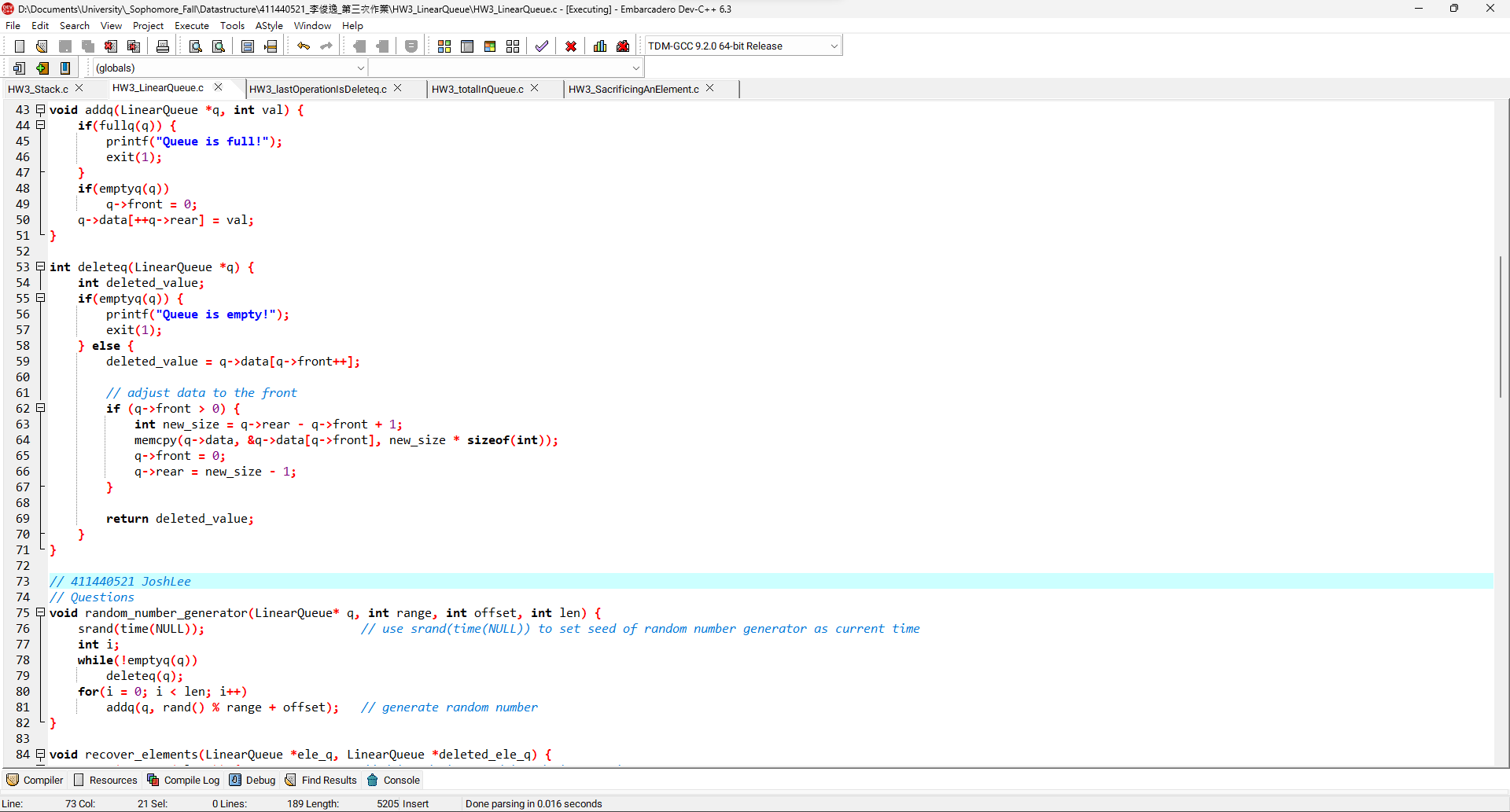
7

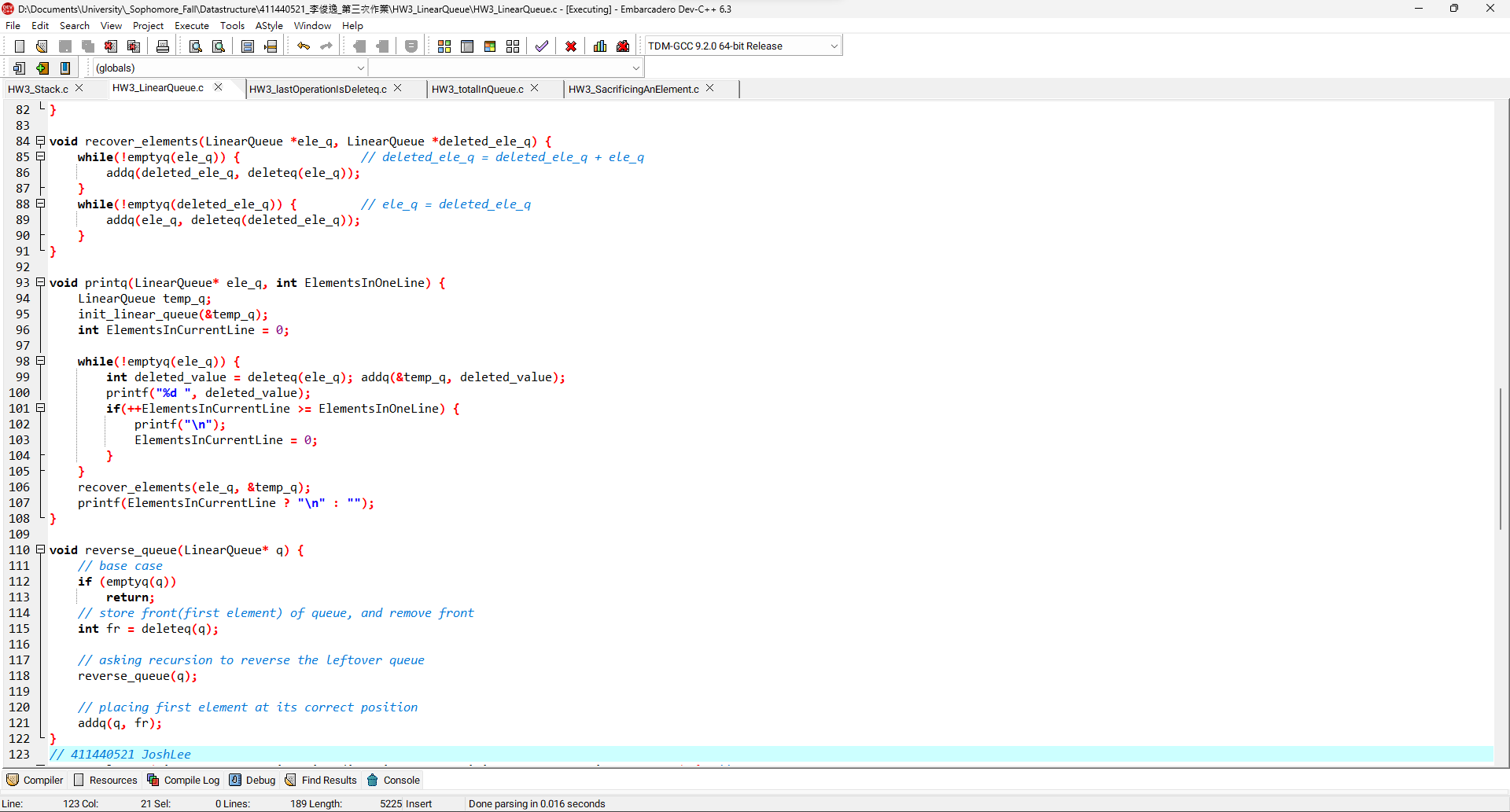
**Stack程式碼:**

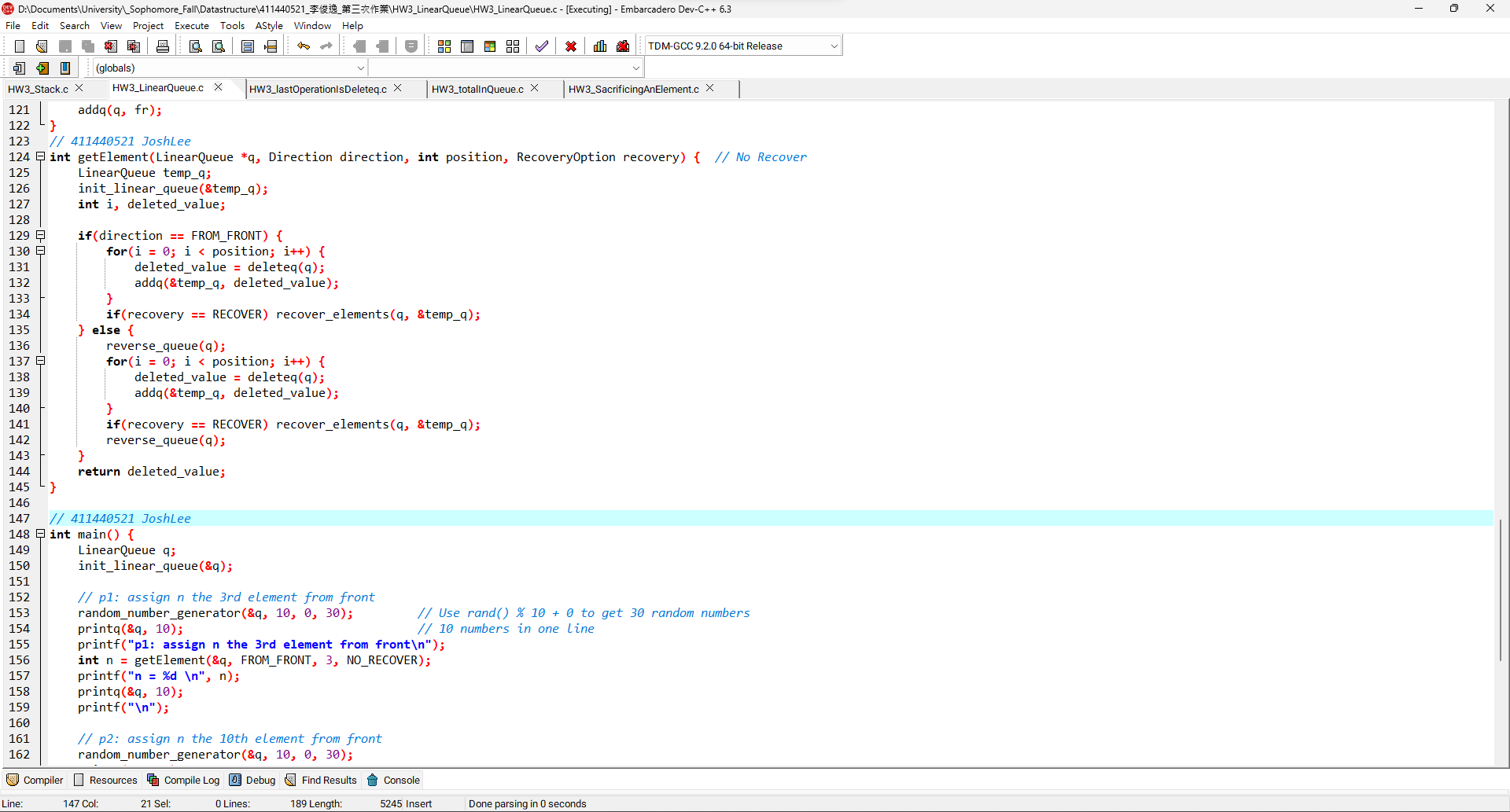
|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Author: 411440521 JoshLee \**  *\* Date: 2023/12/08 \**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  #include <stdio.h>  #include <stdlib.h>  #include <time.h> *// use time() to get current time*  #define SIZE 100  *// Enumeration for flags*  typedef enum {  FROM\_TOP,  FROM\_BOTTOM  } Direction;  typedef enum {  RECOVER,  NO\_RECOVER  } RecoveryOption;  typedef struct {  int data[SIZE];  int top;  } Stack;  void init\_stack(Stack \*s) {  s->top = -1;  }  *// Stack functions*  int fulls(Stack \*s) {  return s->top == SIZE - 1;  }  int emptys(Stack \*s) {  return s->top == -1;  }  void pushs(Stack \*s, int val) {  if(!fulls(s))  s->data[++(s->top)] = val;  else {  printf("Stack is full!");  exit(1);  }  }  int pops(Stack \*s) {  if(!emptys(s))  return s->data[(s->top)--];  printf("Stack is empty!");  exit(2);  }  *// 411440521 JoshLee*  *// Questions*  void random\_number\_generator(Stack \*s, int range, int offset, int len) {  srand(time(NULL)); *// use srand(time(NULL)) to set seed of random number generator as current time*  int i;  while(!emptys(s))  pops(s);  for(i = 0; i < len; i++)  pushs(s, rand() % range + offset); *// generate random number*  }  void recover\_elements(Stack \*s, Stack \*temp\_s) {  while (!emptys(temp\_s)) {  pushs(s, pops(temp\_s));  }  }  void prints(Stack \*ele\_s, int ElementsInOneLine) {  Stack temp\_popped;  init\_stack(&temp\_popped);  int ElementsInCurrentLine = 0;    while(!emptys(ele\_s)) {  int popped\_value = pops(ele\_s);  printf("%d ", popped\_value);  pushs(&temp\_popped, popped\_value);  if(++ElementsInCurrentLine >= ElementsInOneLine) {  printf("\n");  ElementsInCurrentLine = 0;  }  }  recover\_elements(ele\_s, &temp\_popped);  printf(ElementsInCurrentLine ? "\n" : "");  }  void reverse\_stack(Stack \*s) {  Stack s1;  Stack s2;  init\_stack(&s1);  init\_stack(&s2);  while(!emptys(s)) {  pushs(&s1, pops(s));  }  while(!emptys(&s1)) {  pushs(&s2, pops(&s1));  }  while(!emptys(&s2)) {  pushs(s, pops(&s2));  }  }  *// 411440521 JoshLee*  int getElement(Stack \*s, Direction direction, int position, RecoveryOption recovery) {  Stack temp\_s;  init\_stack(&temp\_s);  int i, popped\_value;  if (direction == FROM\_TOP) {  for (i = 0; i < position; i++) {  popped\_value = pops(s);  pushs(&temp\_s, popped\_value);  }  if (recovery == RECOVER)  recover\_elements(s, &temp\_s);  } else {  reverse\_stack(s);  for (i = 0; i < position; i++) {  popped\_value = pops(s);  pushs(&temp\_s, popped\_value);  }  if (recovery == RECOVER)  recover\_elements(s, &temp\_s);  reverse\_stack(s);  }  return popped\_value;  }  *// 411440521 JoshLee*  int main() {  Stack s;  init\_stack(&s);    *// p1: assign m the 3rd element from the top*  random\_number\_generator(&s, 10, 0, 30); *// Use rand() % 10 + 0 to get 30 random numbers*  prints(&s, 10); *// 10 numbers per line*  printf("p1: assign m the 3rd element from the top \n");  int m = getElement(&s, FROM\_TOP, 3, NO\_RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p2: assign m the 3rd element from the top without changing the stack*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p2: assign m the 3rd element from the top without changing the stack \n");  m = getElement(&s, FROM\_TOP, 3, RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p3: assign m the 10th element from the top*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p3: assign m the 10th element from the top \n");  m = getElement(&s, FROM\_TOP, 10, NO\_RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p4: assign m the 10th element from the top without changing the stack*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p4: assign m the 10th element from the top without changing the stack \n");  m = getElement(&s, FROM\_TOP, 10, RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");  *// 411440521 JoshLee*  *// p5: assign m the bottom element*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p5: assign m the bottom element \n");  m = getElement(&s, FROM\_BOTTOM, 1, NO\_RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p6: assign m the bottom element without changing the stack*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p6: assign m the bottom element without changing the stack \n");  m = getElement(&s, FROM\_BOTTOM, 1, RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p7: assign m the 2nd element from the bottom*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p7: assign m the 2nd element from the bottom \n");  m = getElement(&s, FROM\_BOTTOM, 2, NO\_RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p8: assign m the 2nd element from the bottom without changing the stack*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p8: assign m the 2nd element from the bottom without changing the stack \n");  m = getElement(&s, FROM\_BOTTOM, 2, RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p9: assign m the 3rd element from the bottom*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p9: assign m the 3rd element from the bottom \n");  m = getElement(&s, FROM\_BOTTOM, 3, NO\_RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p10: assign m the 3rd element from the bottom without changing the stack*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p10: assign m the 3rd element from the bottom without changing the stack \n");  m = getElement(&s, FROM\_BOTTOM, 3, RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p11: assign m the 4th element from the bottom*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p11: assign m the 4th element from the bottom \n");  m = getElement(&s, FROM\_BOTTOM, 4, NO\_RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");    *// p12: assign m the 4th element from the bottom without changing the stack*  random\_number\_generator(&s, 10, 0, 30);  prints(&s, 10);  printf("p12: assign m the 4th element from the bottom without changing the stack \n");  m = getElement(&s, FROM\_BOTTOM, 4, RECOVER);  printf("m = %d \n", m);  prints(&s, 10);  printf("\n");  *// 411440521 JoshLee*  return 0;  } |

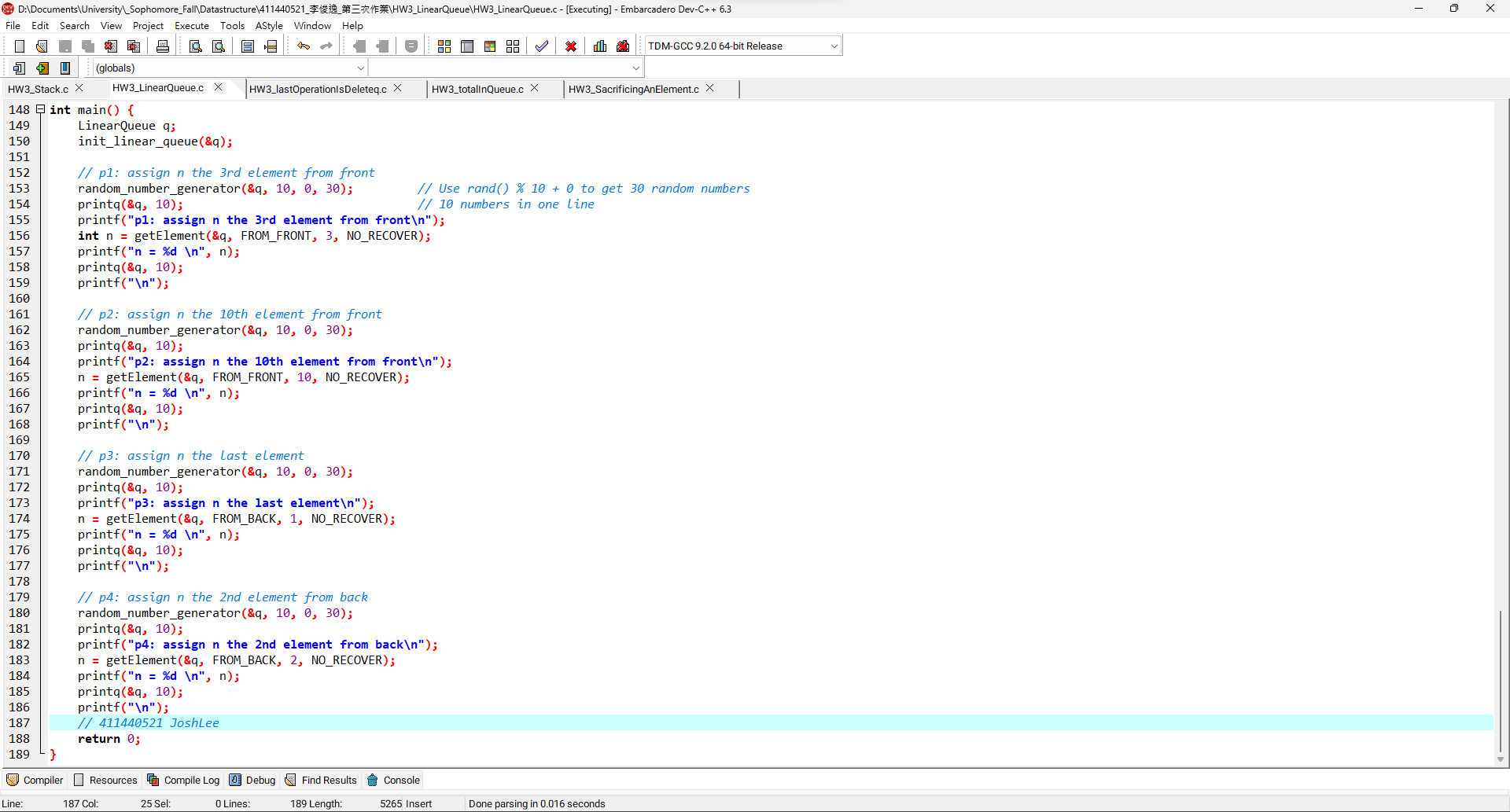
1

2

3

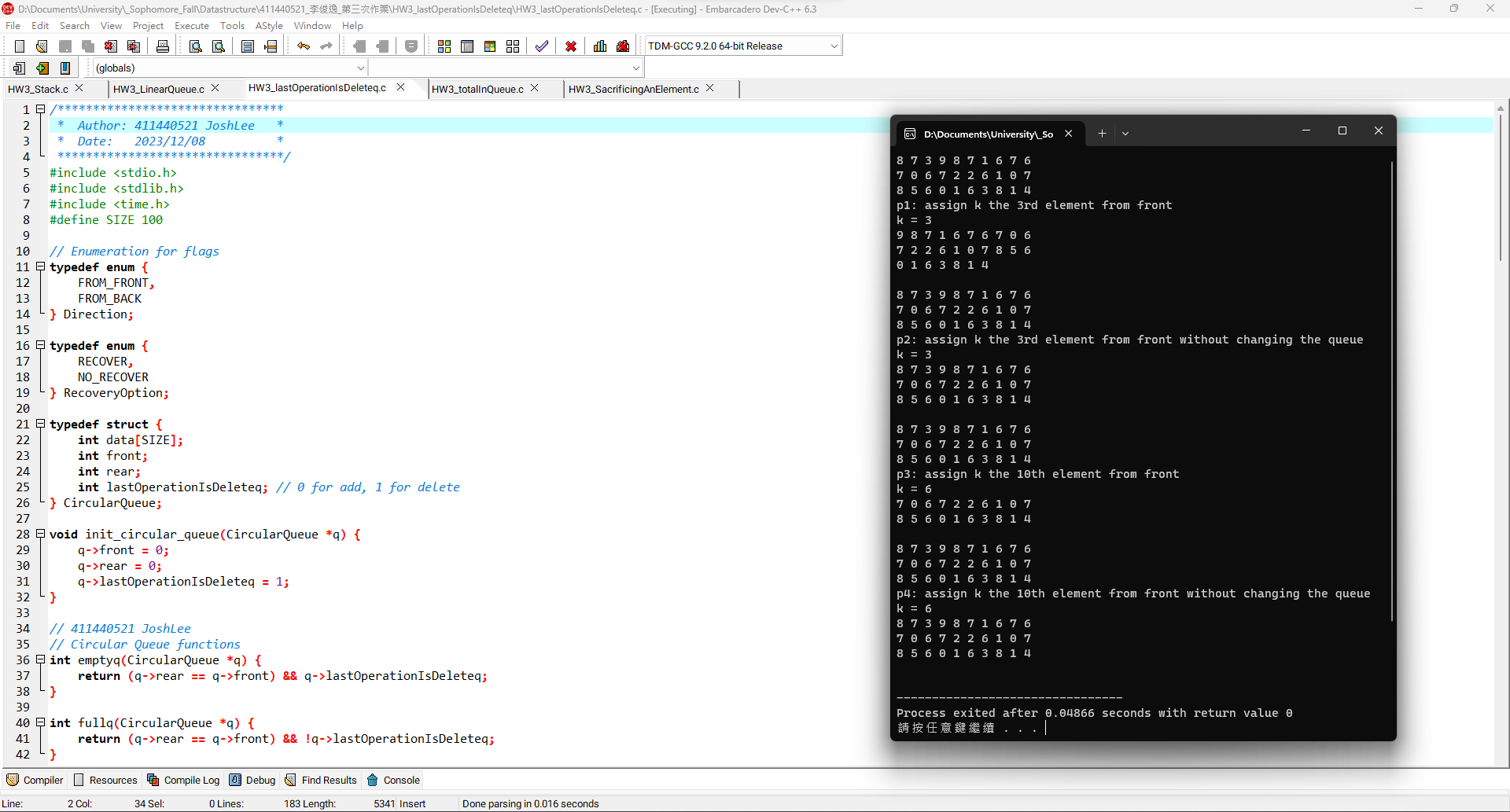
4

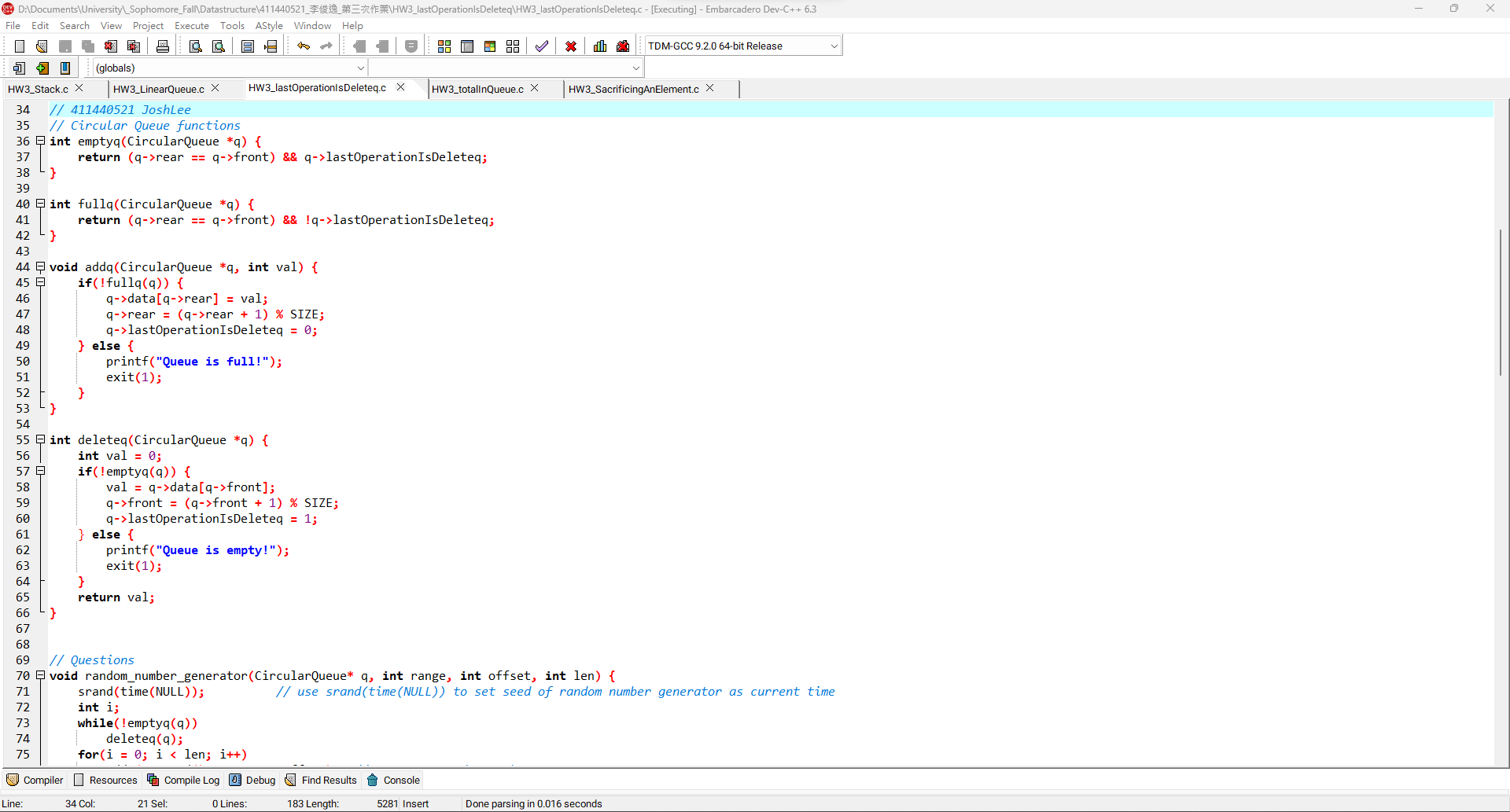
5

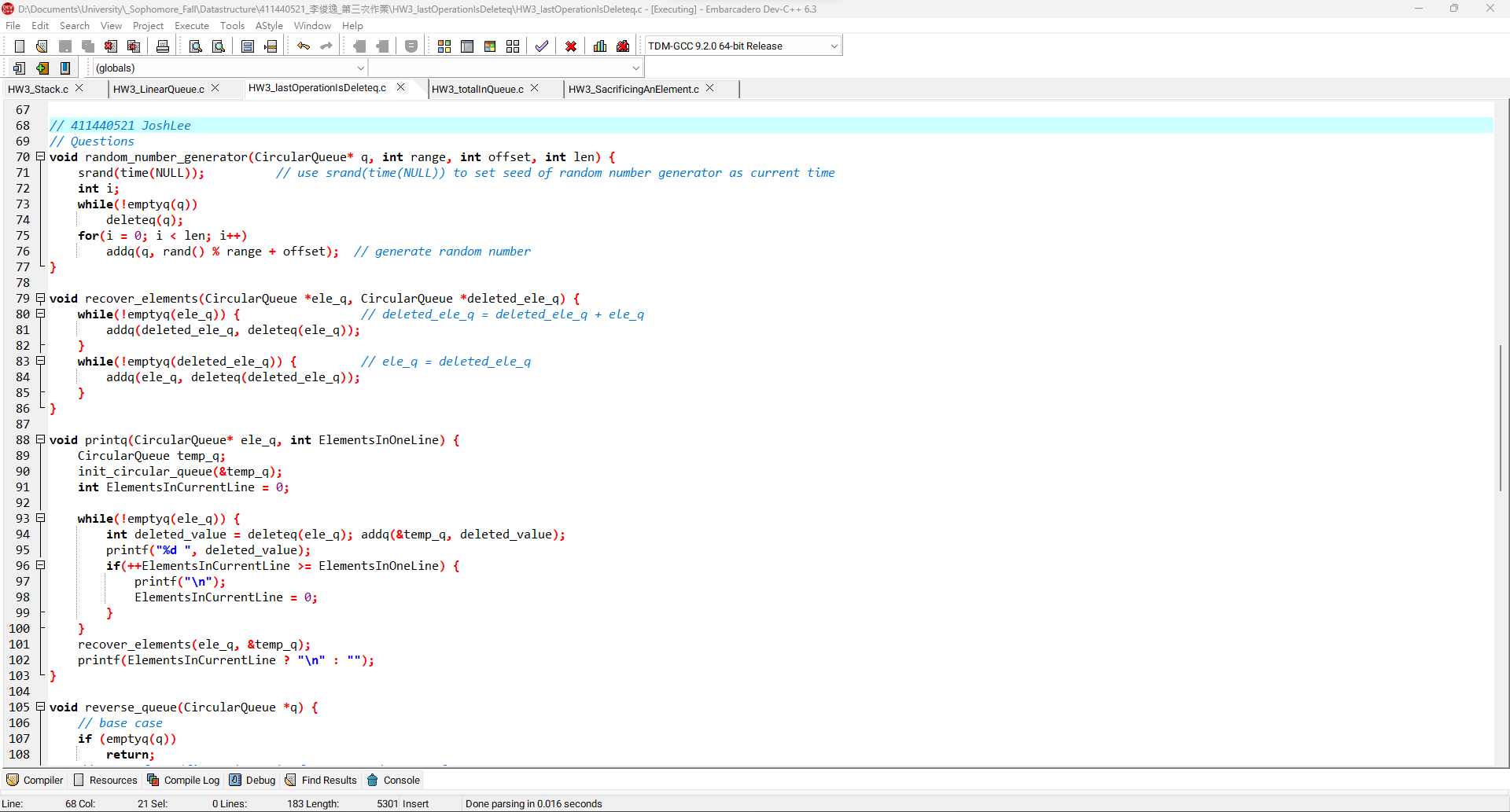
6

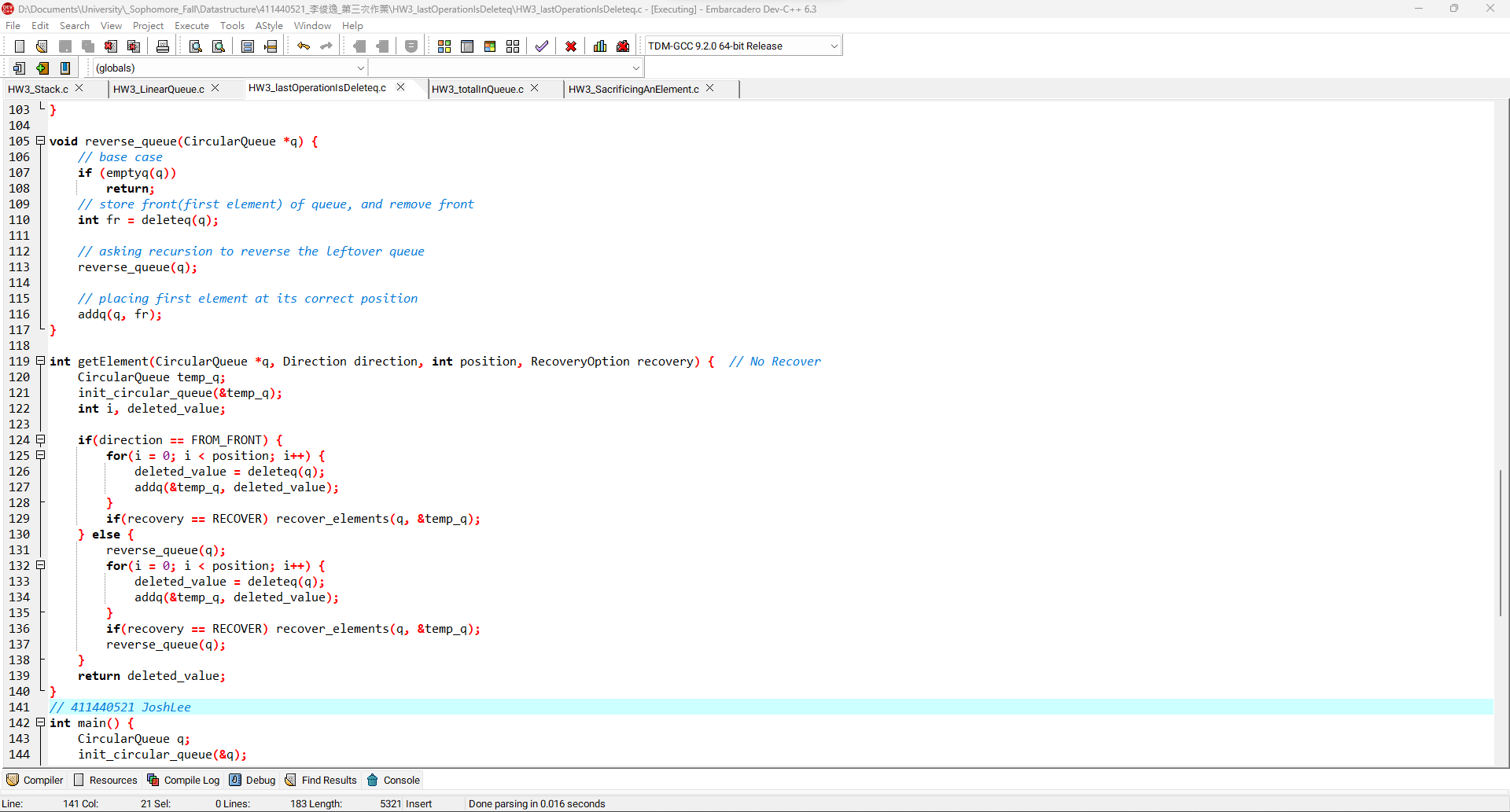
**LinearQueue程式碼:**

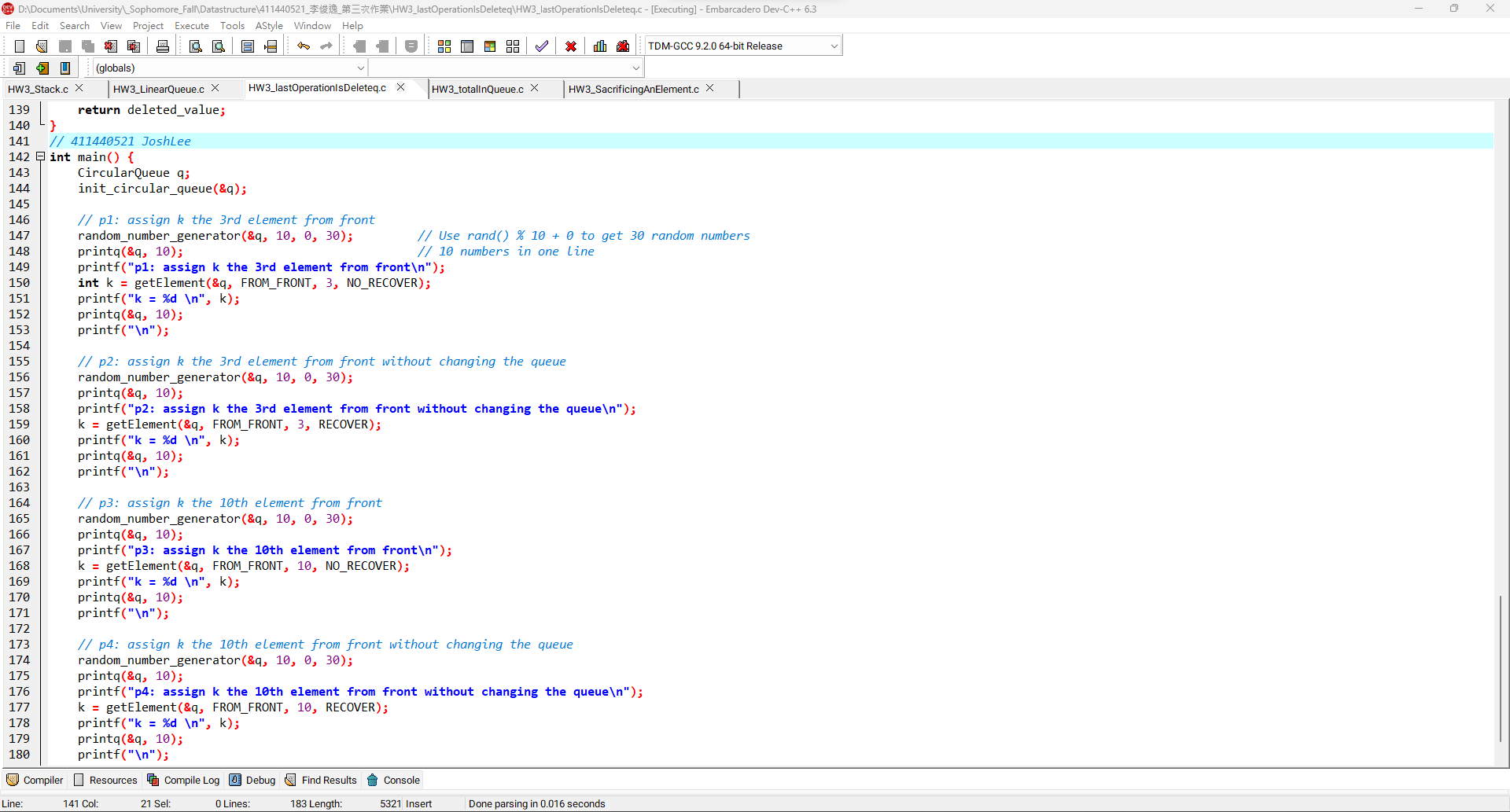
|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Author: 411440521 JoshLee \**  *\* Date: 2023/12/08 \**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  #include <stdio.h>  #include <stdlib.h>  #include <string.h> *// memcpy()*  #include <time.h>  #define SIZE 100  *// Enumeration for flags*  typedef enum {  FROM\_FRONT,  FROM\_BACK  } Direction;  typedef enum {  RECOVER,  NO\_RECOVER  } RecoveryOption;  typedef struct {  int data[SIZE];  int front;  int rear;  } LinearQueue;  void init\_linear\_queue(LinearQueue \*q){  q->front = -1;  q->rear = -1;  }  *// Linear Queue functions*  int emptyq(LinearQueue \*q) {  return (q->front == -1) || (q->front > q->rear);  }  int fullq(LinearQueue \*q) {  return q->rear == SIZE - 1;  }  void addq(LinearQueue \*q, int val) {  if(fullq(q)) {  printf("Queue is full!");  exit(1);  }  if(emptyq(q))  q->front = 0;  q->data[++q->rear] = val;  }  int deleteq(LinearQueue \*q) {  int deleted\_value;  if(emptyq(q)) {  printf("Queue is empty!");  exit(1);  } else {  deleted\_value = q->data[q->front++];    *// adjust data to the front*  if (q->front > 0) {  int new\_size = q->rear - q->front + 1;  memcpy(q->data, &q->data[q->front], new\_size \* sizeof(int));  q->front = 0;  q->rear = new\_size - 1;  }  return deleted\_value;  }  }  *// 411440521 JoshLee*  *// Questions*  void random\_number\_generator(LinearQueue\* q, int range, int offset, int len) {  srand(time(NULL)); *// use srand(time(NULL)) to set seed of random number generator as current time*  int i;  while(!emptyq(q))  deleteq(q);  for(i = 0; i < len; i++)  addq(q, rand() % range + offset); *// generate random number*  }  void recover\_elements(LinearQueue \*ele\_q, LinearQueue \*deleted\_ele\_q) {  while(!emptyq(ele\_q)) { *// deleted\_ele\_q = deleted\_ele\_q + ele\_q*  addq(deleted\_ele\_q, deleteq(ele\_q));  }  while(!emptyq(deleted\_ele\_q)) { *// ele\_q = deleted\_ele\_q*  addq(ele\_q, deleteq(deleted\_ele\_q));  }  }  void printq(LinearQueue\* ele\_q, int ElementsInOneLine) {  LinearQueue temp\_q;  init\_linear\_queue(&temp\_q);  int ElementsInCurrentLine = 0;  while(!emptyq(ele\_q)) {  int deleted\_value = deleteq(ele\_q); addq(&temp\_q, deleted\_value);  printf("%d ", deleted\_value);  if(++ElementsInCurrentLine >= ElementsInOneLine) {  printf("\n");  ElementsInCurrentLine = 0;  }  }  recover\_elements(ele\_q, &temp\_q);  printf(ElementsInCurrentLine ? "\n" : "");  }  void reverse\_queue(LinearQueue\* q) {  *// base case*  if (emptyq(q))  return;  *// store front(first element) of queue, and remove front*  int fr = deleteq(q);    *// asking recursion to reverse the leftover queue*  reverse\_queue(q);    *// placing first element at its correct position*  addq(q, fr);  }  *// 411440521 JoshLee*  int getElement(LinearQueue \*q, Direction direction, int position, RecoveryOption recovery) { *// No Recover*  LinearQueue temp\_q;  init\_linear\_queue(&temp\_q);  int i, deleted\_value;  if(direction == FROM\_FRONT) {  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  } else {  reverse\_queue(q);  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  reverse\_queue(q);  }  return deleted\_value;  }  *// 411440521 JoshLee*  int main() {  LinearQueue q;  init\_linear\_queue(&q);    *// p1: assign n the 3rd element from front*  random\_number\_generator(&q, 10, 0, 30); *// Use rand() % 10 + 0 to get 30 random numbers*  printq(&q, 10); *// 10 numbers in one line*  printf("p1: assign n the 3rd element from front\n");  int n = getElement(&q, FROM\_FRONT, 3, NO\_RECOVER);  printf("n = %d \n", n);  printq(&q, 10);  printf("\n");    *// p2: assign n the 10th element from front*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p2: assign n the 10th element from front\n");  n = getElement(&q, FROM\_FRONT, 10, NO\_RECOVER);  printf("n = %d \n", n);  printq(&q, 10);  printf("\n");    *// p3: assign n the last element*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p3: assign n the last element\n");  n = getElement(&q, FROM\_BACK, 1, NO\_RECOVER);  printf("n = %d \n", n);  printq(&q, 10);  printf("\n");    *// p4: assign n the 2nd element from back*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p4: assign n the 2nd element from back\n");  n = getElement(&q, FROM\_BACK, 2, NO\_RECOVER);  printf("n = %d \n", n);  printq(&q, 10);  printf("\n");  *// 411440521 JoshLee*  return 0;  } |

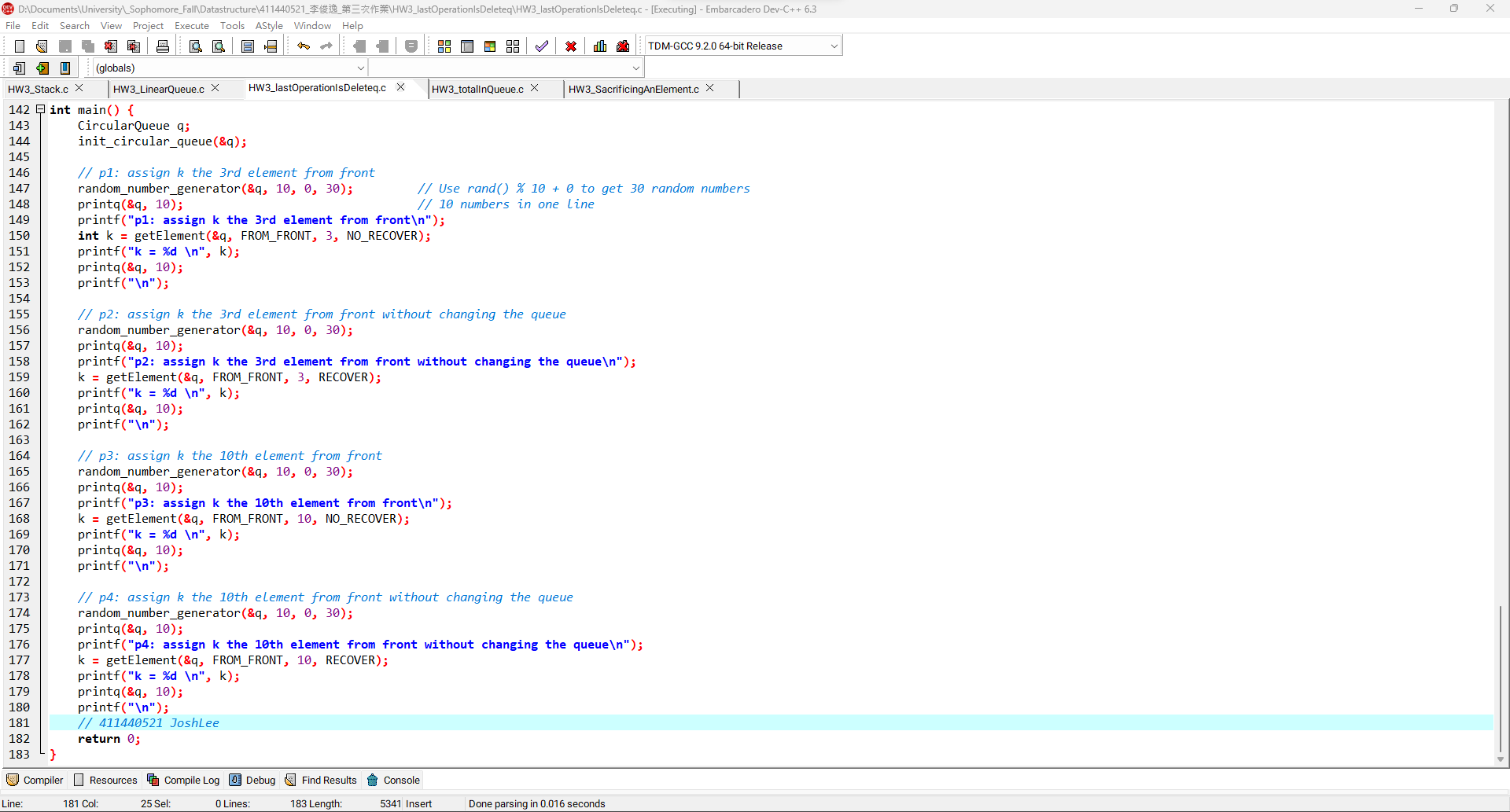
1

2

3

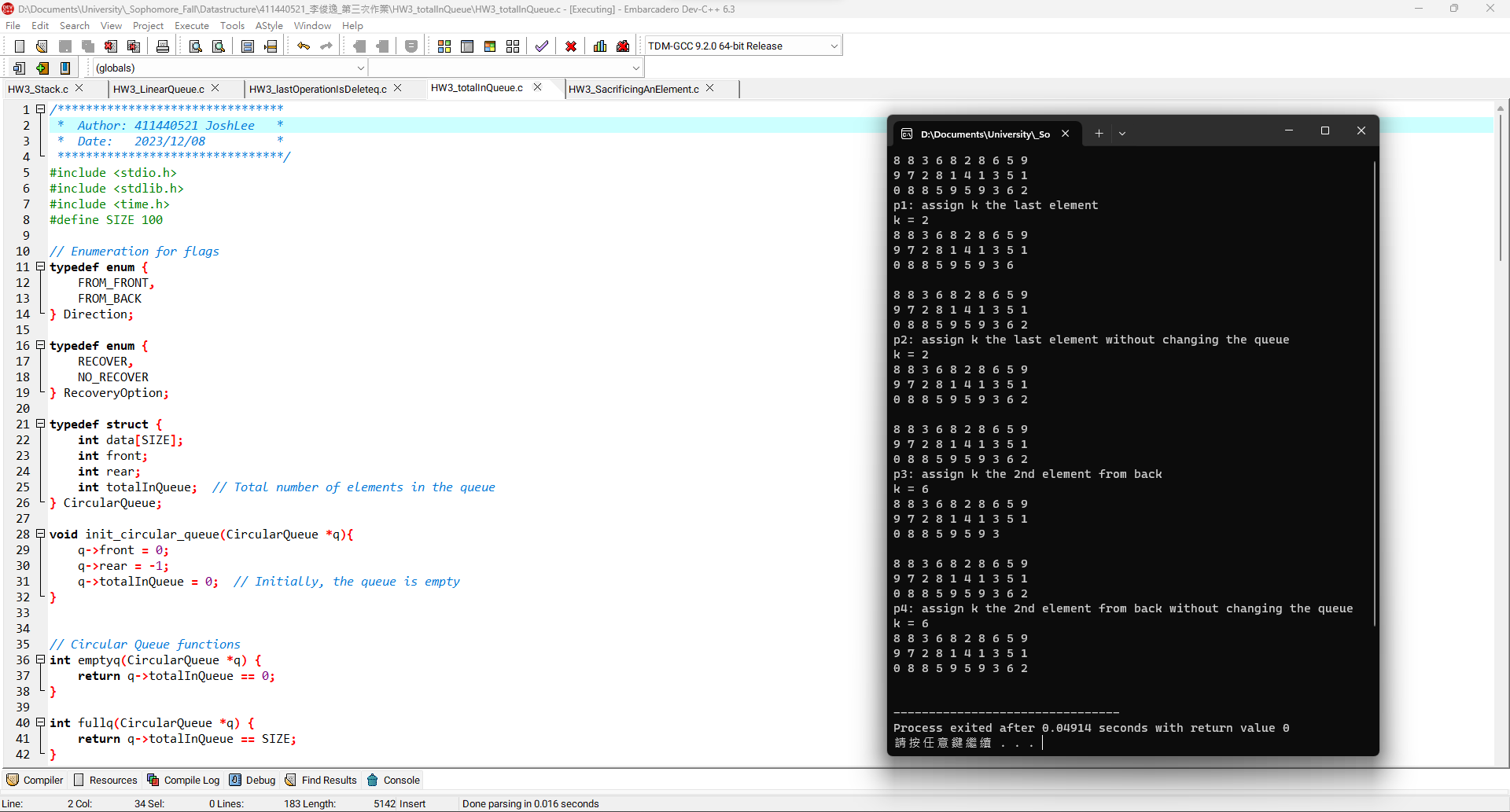
4

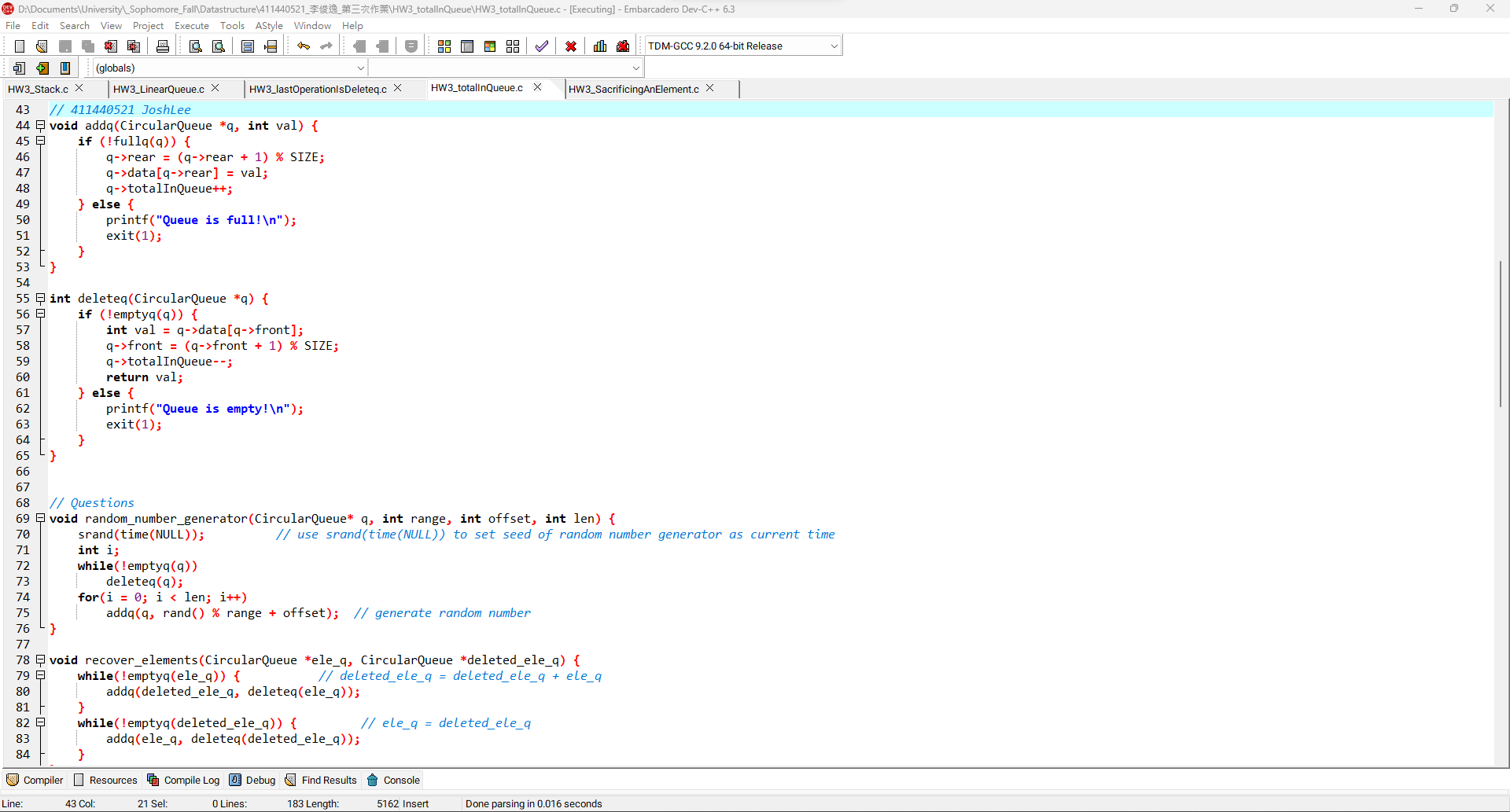
5

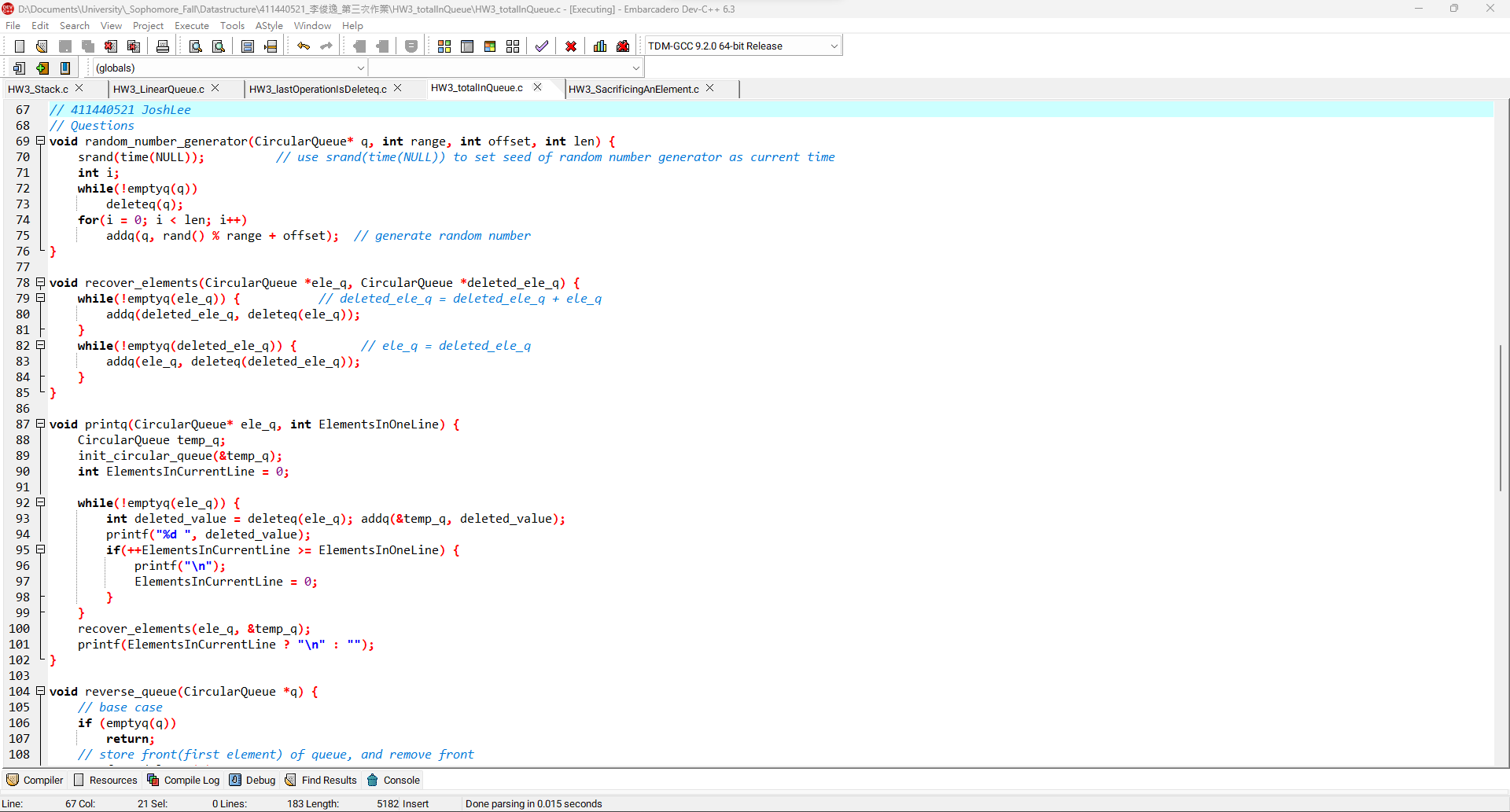
6

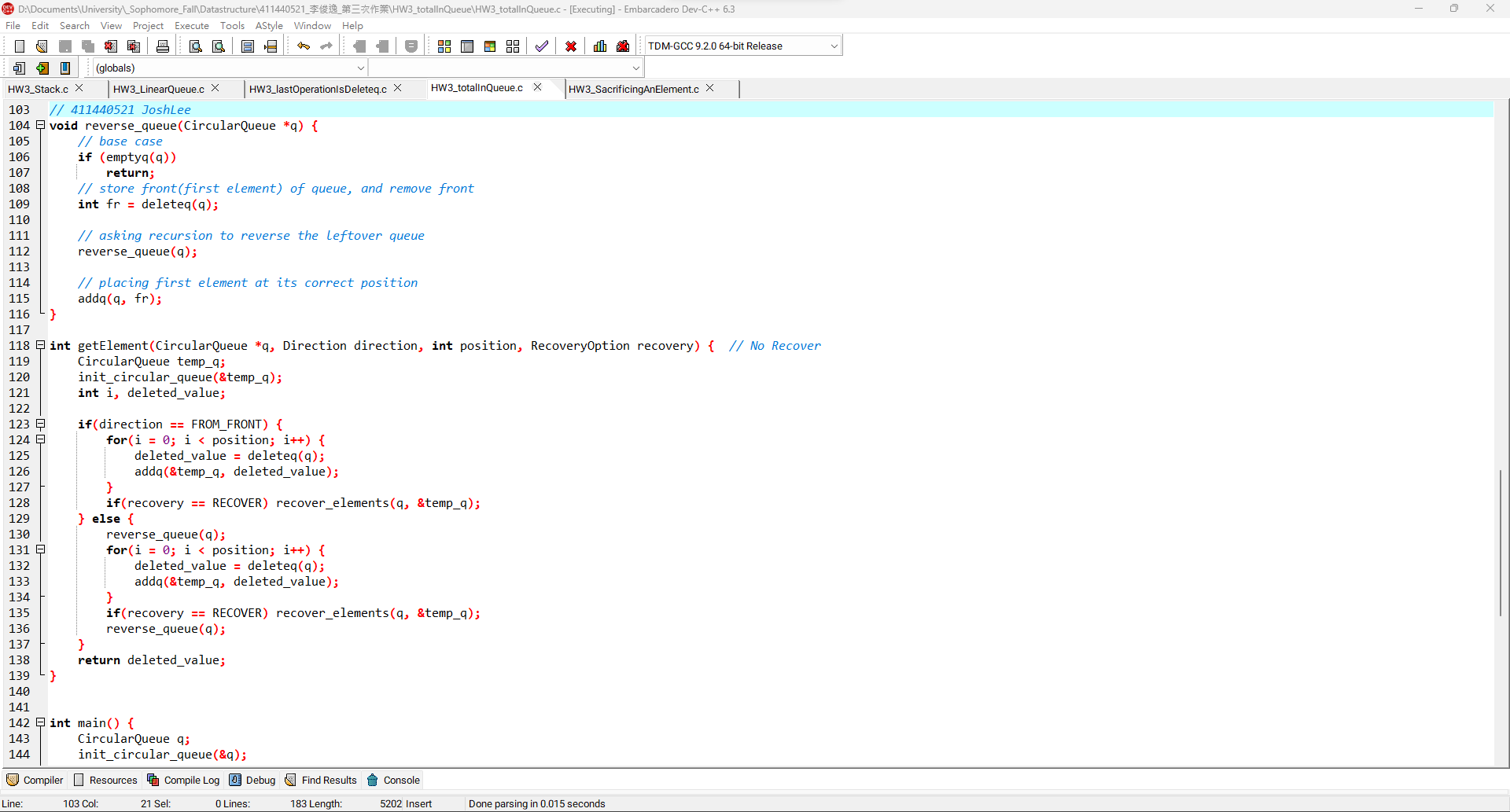
**lastOperationIsDeleteq程式碼:**

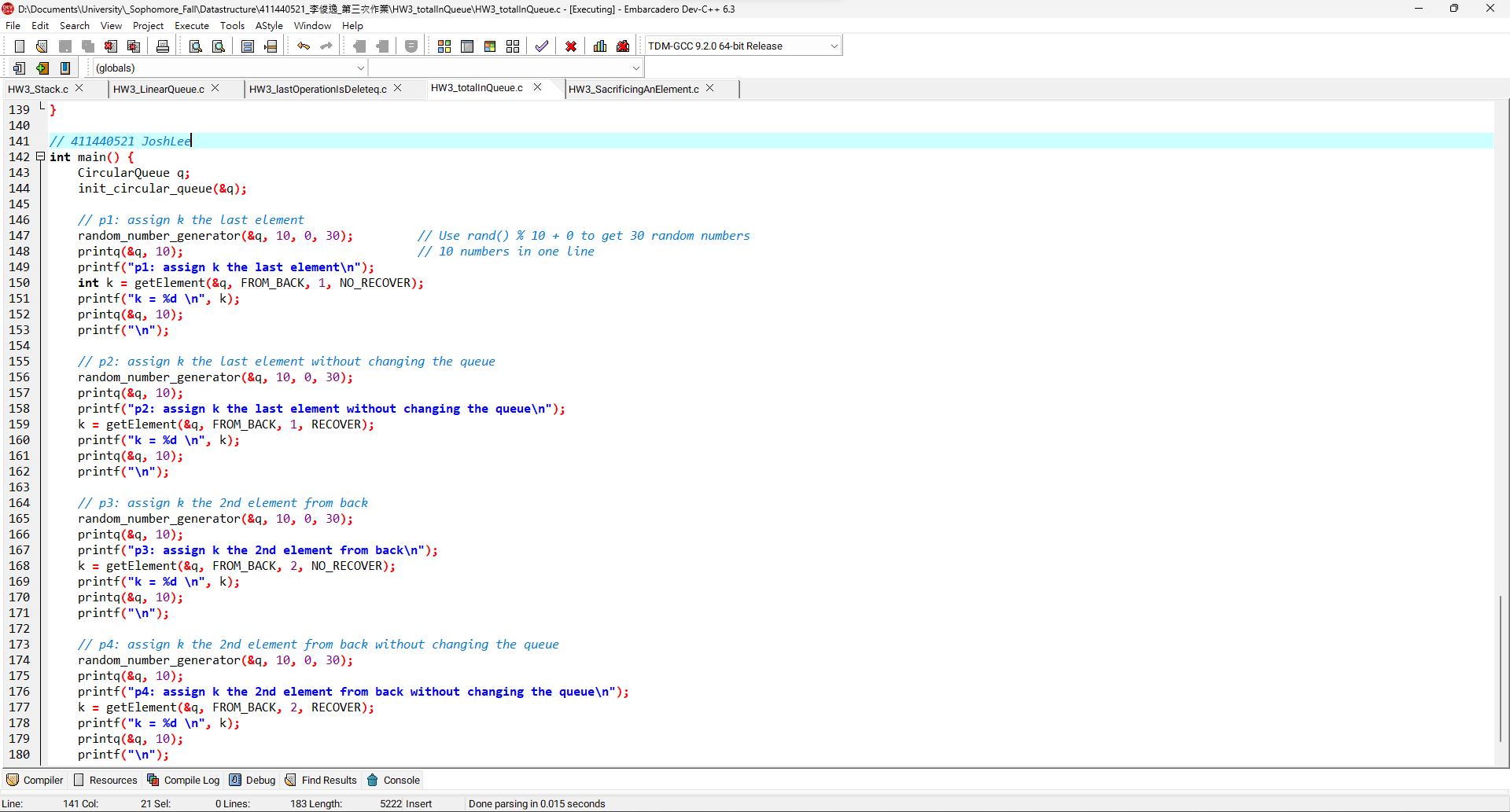
|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Author: 411440521 JoshLee \**  *\* Date: 2023/12/08 \**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define SIZE 100  *// Enumeration for flags*  typedef enum {  FROM\_FRONT,  FROM\_BACK  } Direction;  typedef enum {  RECOVER,  NO\_RECOVER  } RecoveryOption;  typedef struct {  int data[SIZE];  int front;  int rear;  int lastOperationIsDeleteq; *// 0 for add, 1 for delete*  } CircularQueue;  void init\_circular\_queue(CircularQueue \*q) {  q->front = 0;  q->rear = 0;  q->lastOperationIsDeleteq = 1;  }  *// 411440521 JoshLee*  *// Circular Queue functions*  int emptyq(CircularQueue \*q) {  return (q->rear == q->front) && q->lastOperationIsDeleteq;  }  int fullq(CircularQueue \*q) {  return (q->rear == q->front) && !q->lastOperationIsDeleteq;  }  void addq(CircularQueue \*q, int val) {  if(!fullq(q)) {  q->data[q->rear] = val;  q->rear = (q->rear + 1) % SIZE;  q->lastOperationIsDeleteq = 0;  } else {  printf("Queue is full!");  exit(1);  }  }  int deleteq(CircularQueue \*q) {  int val = 0;  if(!emptyq(q)) {  val = q->data[q->front];  q->front = (q->front + 1) % SIZE;  q->lastOperationIsDeleteq = 1;  } else {  printf("Queue is empty!");  exit(1);  }  return val;  }  *// 411440521 JoshLee*  *// Questions*  void random\_number\_generator(CircularQueue\* q, int range, int offset, int len) {  srand(time(NULL)); *// use srand(time(NULL)) to set seed of random number generator as current time*  int i;  while(!emptyq(q))  deleteq(q);  for(i = 0; i < len; i++)  addq(q, rand() % range + offset); *// generate random number*  }  void recover\_elements(CircularQueue \*ele\_q, CircularQueue \*deleted\_ele\_q) {  while(!emptyq(ele\_q)) { *// deleted\_ele\_q = deleted\_ele\_q + ele\_q*  addq(deleted\_ele\_q, deleteq(ele\_q));  }  while(!emptyq(deleted\_ele\_q)) { *// ele\_q = deleted\_ele\_q*  addq(ele\_q, deleteq(deleted\_ele\_q));  }  }  void printq(CircularQueue\* ele\_q, int ElementsInOneLine) {  CircularQueue temp\_q;  init\_circular\_queue(&temp\_q);  int ElementsInCurrentLine = 0;  while(!emptyq(ele\_q)) {  int deleted\_value = deleteq(ele\_q); addq(&temp\_q, deleted\_value);  printf("%d ", deleted\_value);  if(++ElementsInCurrentLine >= ElementsInOneLine) {  printf("\n");  ElementsInCurrentLine = 0;  }  }  recover\_elements(ele\_q, &temp\_q);  printf(ElementsInCurrentLine ? "\n" : "");  }  void reverse\_queue(CircularQueue \*q) {  *// base case*  if (emptyq(q))  return;  *// store front(first element) of queue, and remove front*  int fr = deleteq(q);    *// asking recursion to reverse the leftover queue*  reverse\_queue(q);    *// placing first element at its correct position*  addq(q, fr);  }  int getElement(CircularQueue \*q, Direction direction, int position, RecoveryOption recovery) { *// No Recover*  CircularQueue temp\_q;  init\_circular\_queue(&temp\_q);  int i, deleted\_value;  if(direction == FROM\_FRONT) {  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  } else {  reverse\_queue(q);  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  reverse\_queue(q);  }  return deleted\_value;  }  *// 411440521 JoshLee*  int main() {  CircularQueue q;  init\_circular\_queue(&q);  *// p1: assign k the 3rd element from front*  random\_number\_generator(&q, 10, 0, 30); *// Use rand() % 10 + 0 to get 30 random numbers*  printq(&q, 10); *// 10 numbers in one line*  printf("p1: assign k the 3rd element from front\n");  int k = getElement(&q, FROM\_FRONT, 3, NO\_RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p2: assign k the 3rd element from front without changing the queue*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p2: assign k the 3rd element from front without changing the queue\n");  k = getElement(&q, FROM\_FRONT, 3, RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p3: assign k the 10th element from front*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p3: assign k the 10th element from front\n");  k = getElement(&q, FROM\_FRONT, 10, NO\_RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p4: assign k the 10th element from front without changing the queue*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p4: assign k the 10th element from front without changing the queue\n");  k = getElement(&q, FROM\_FRONT, 10, RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");  *// 411440521 JoshLee*  return 0;  } |

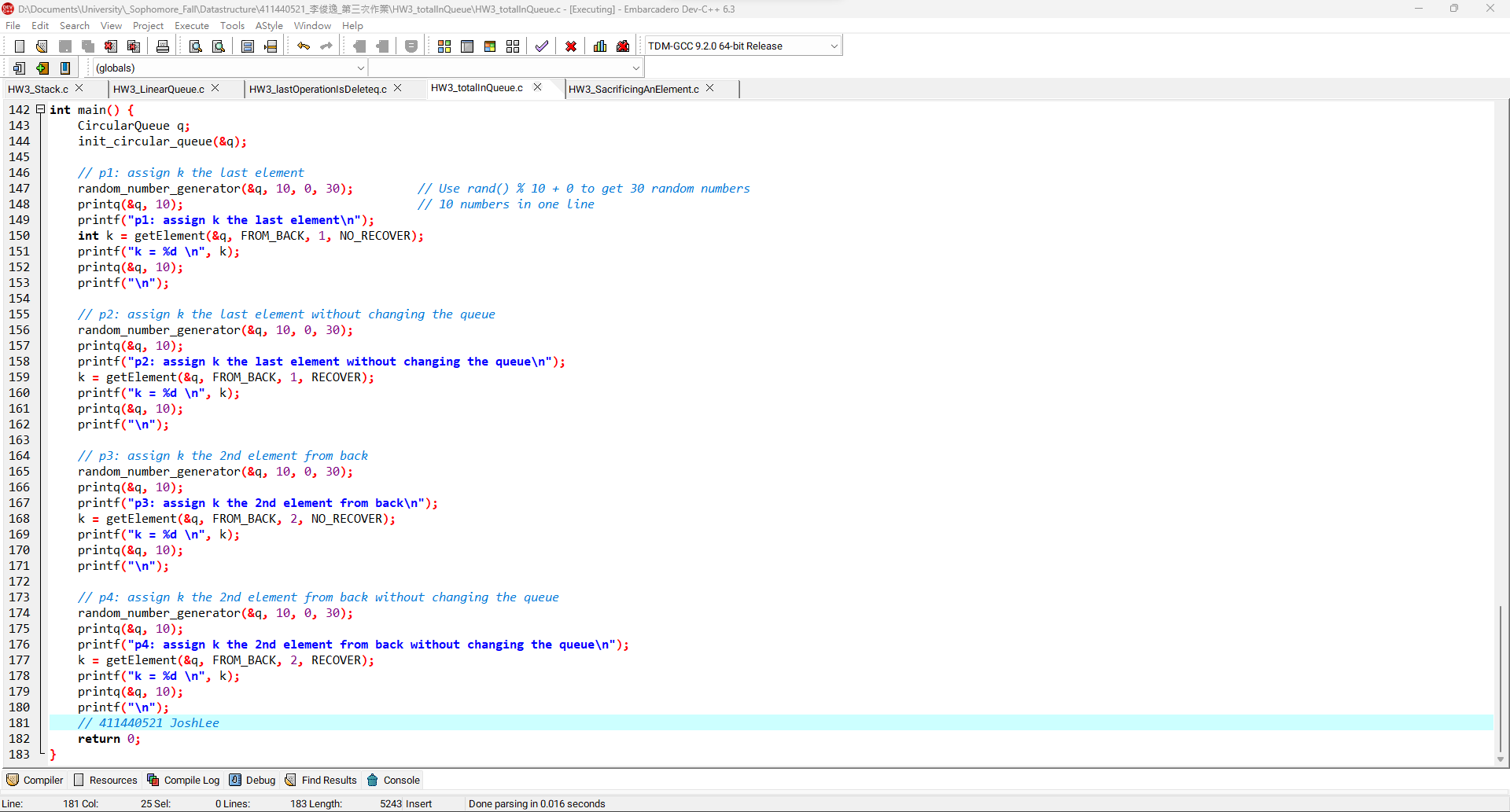
1

2

3

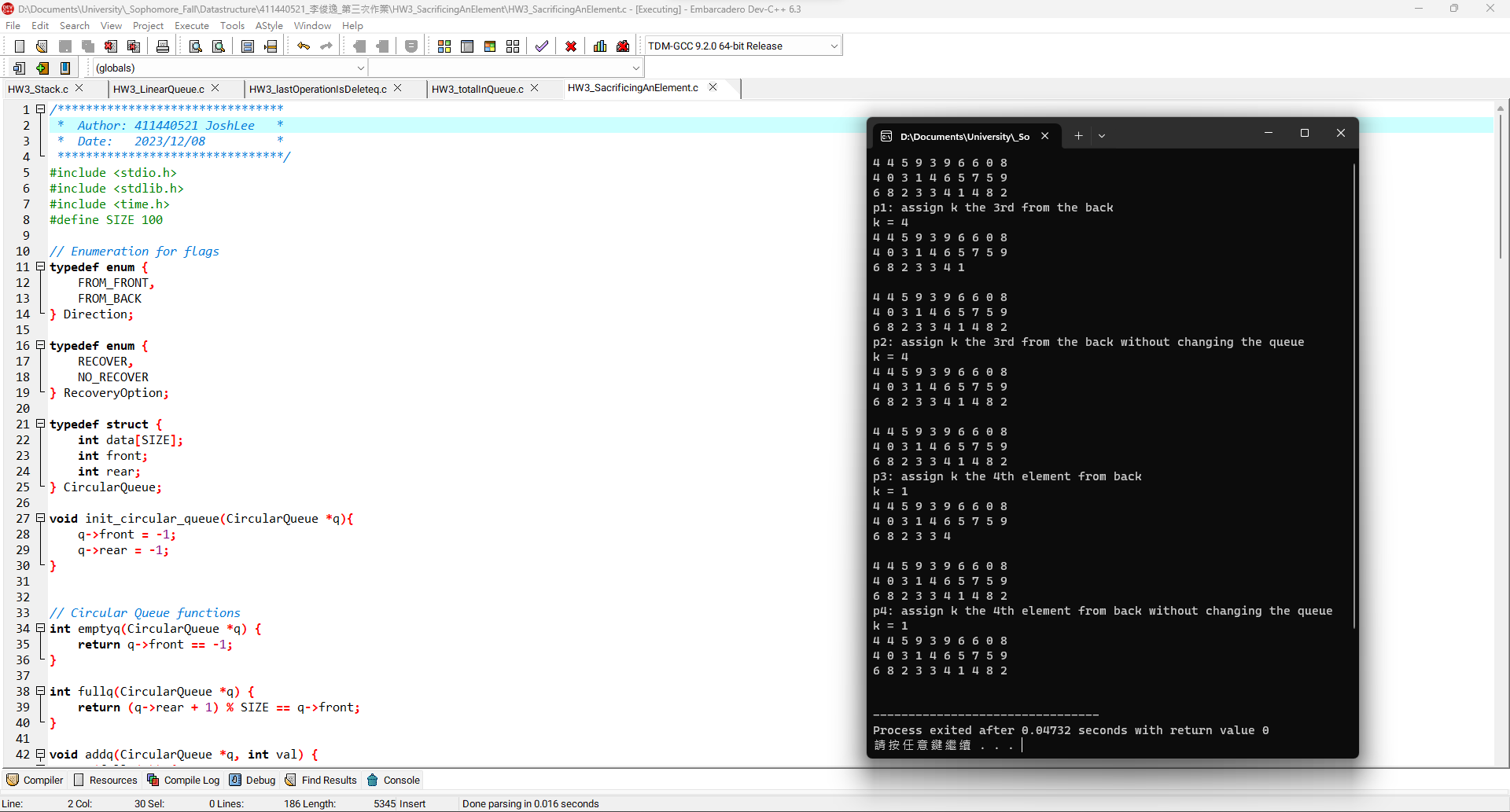
4

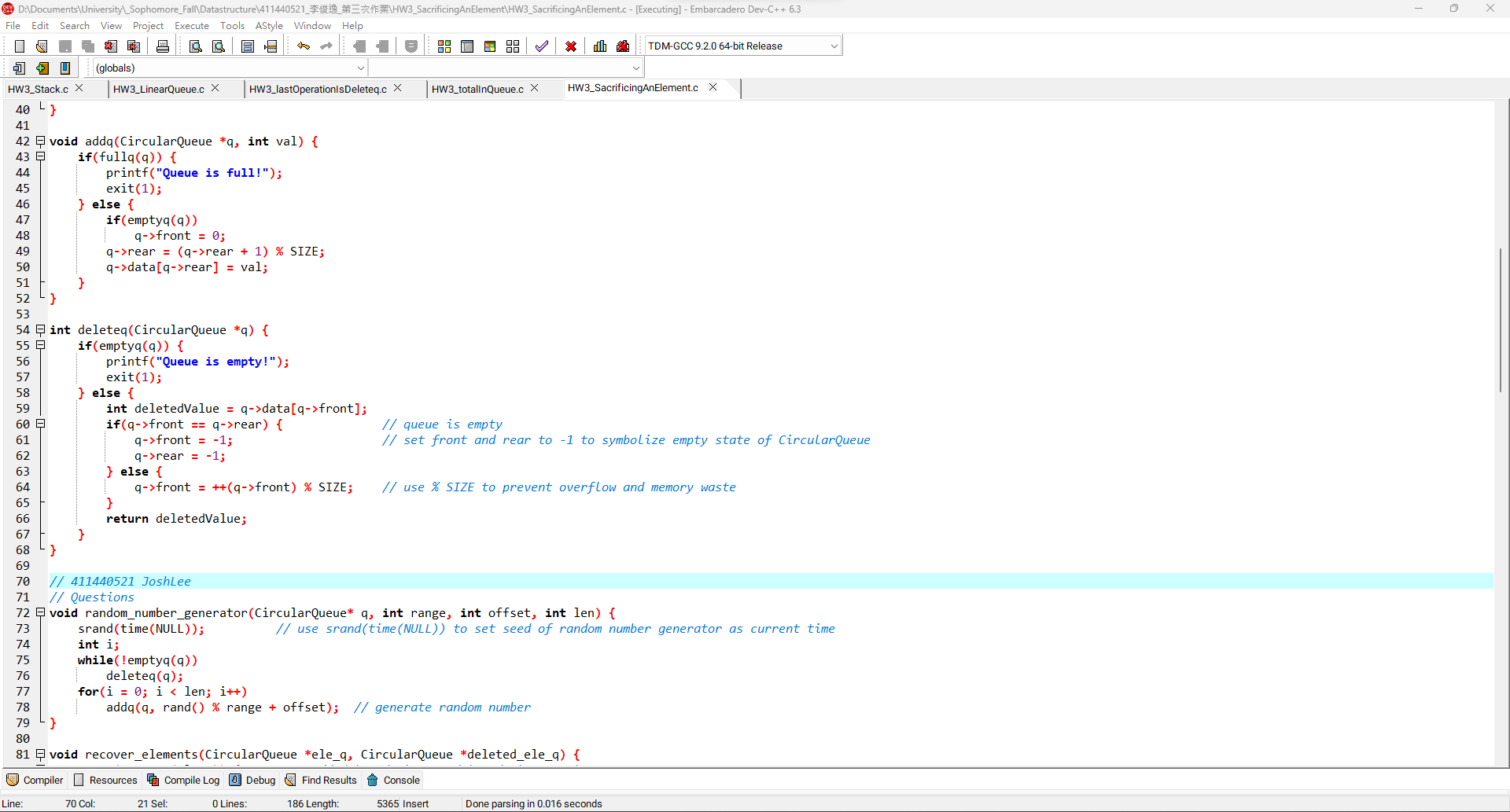
5

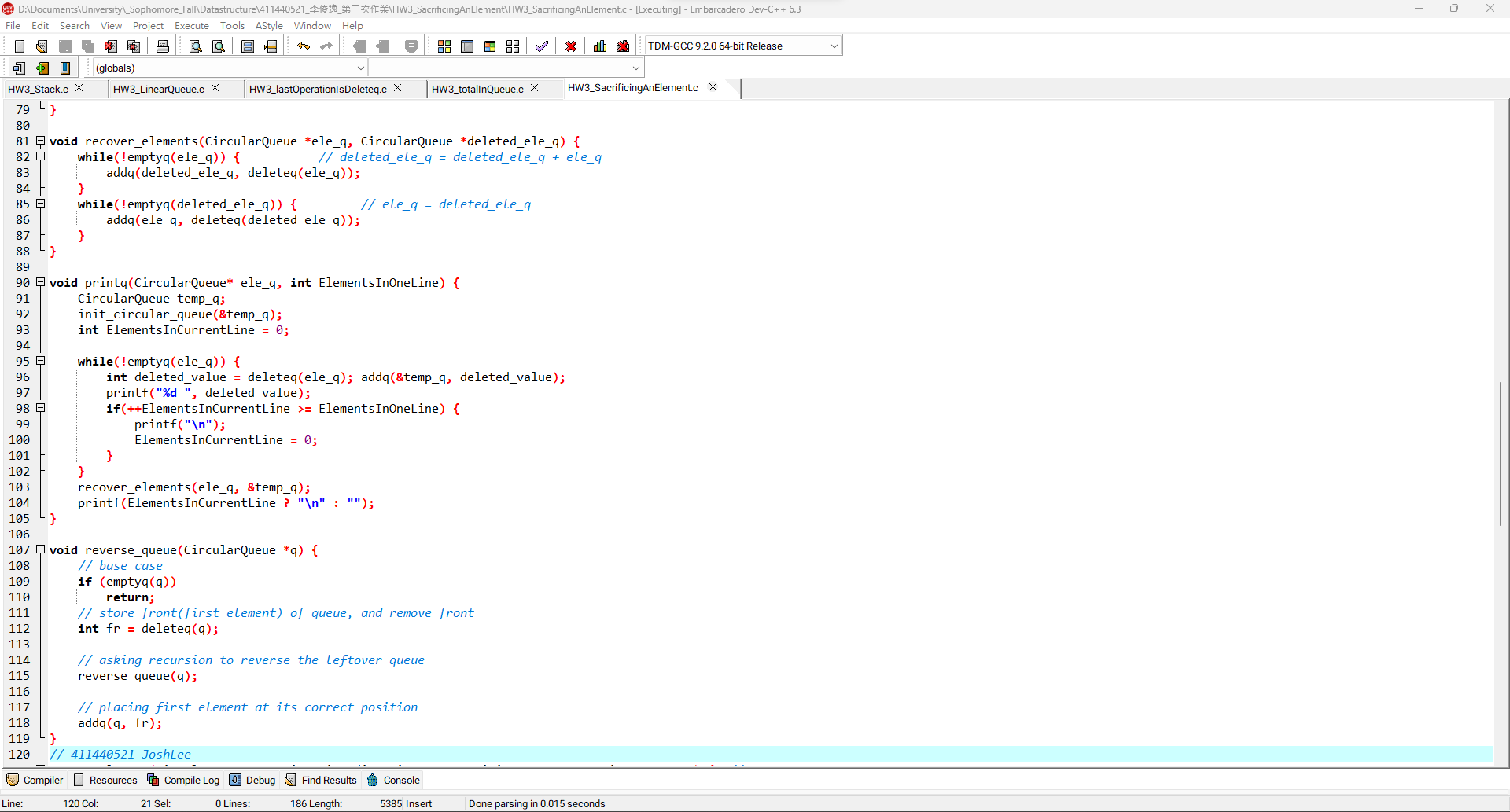
6

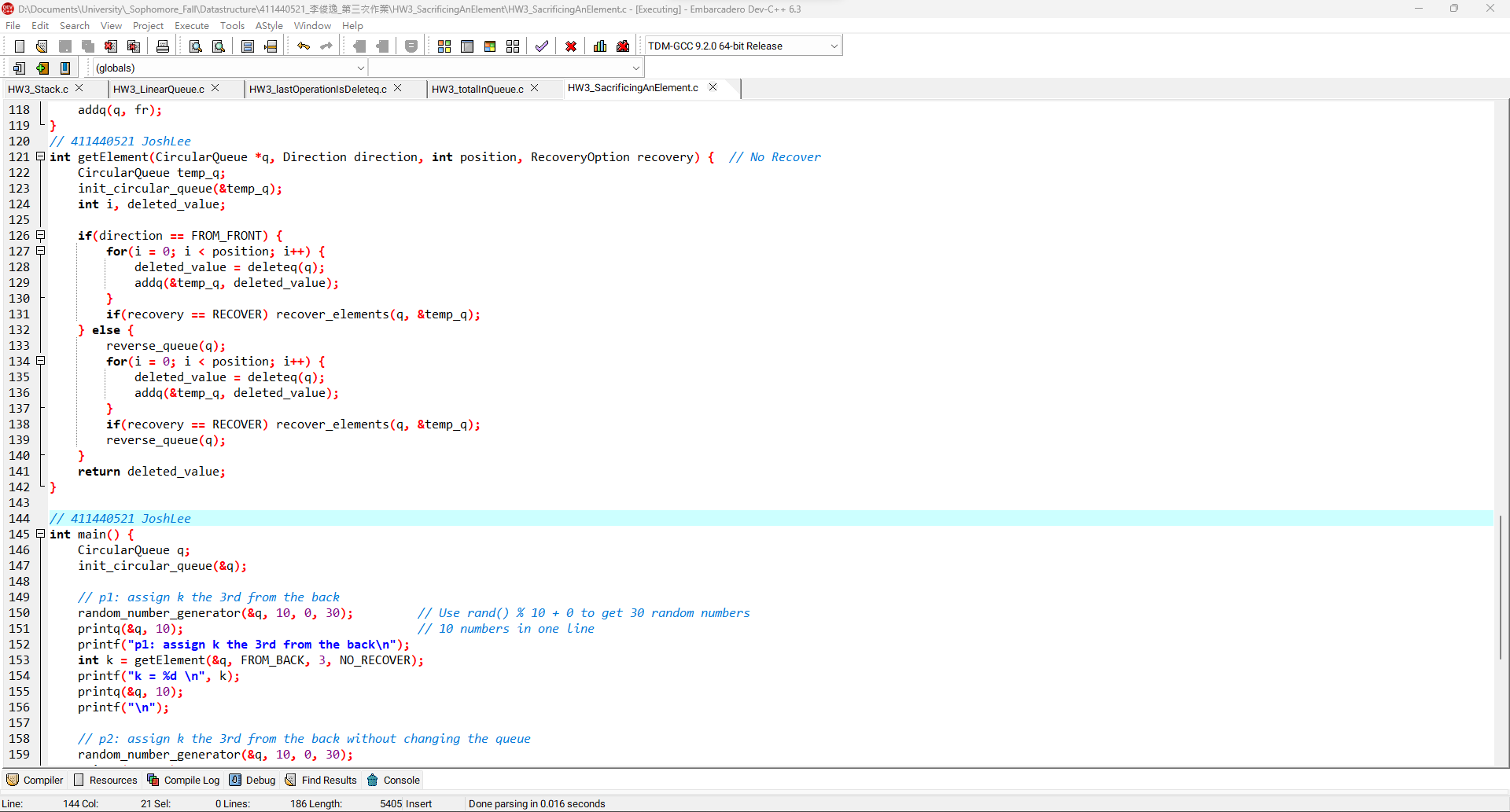
**totalInQueue程式碼:**

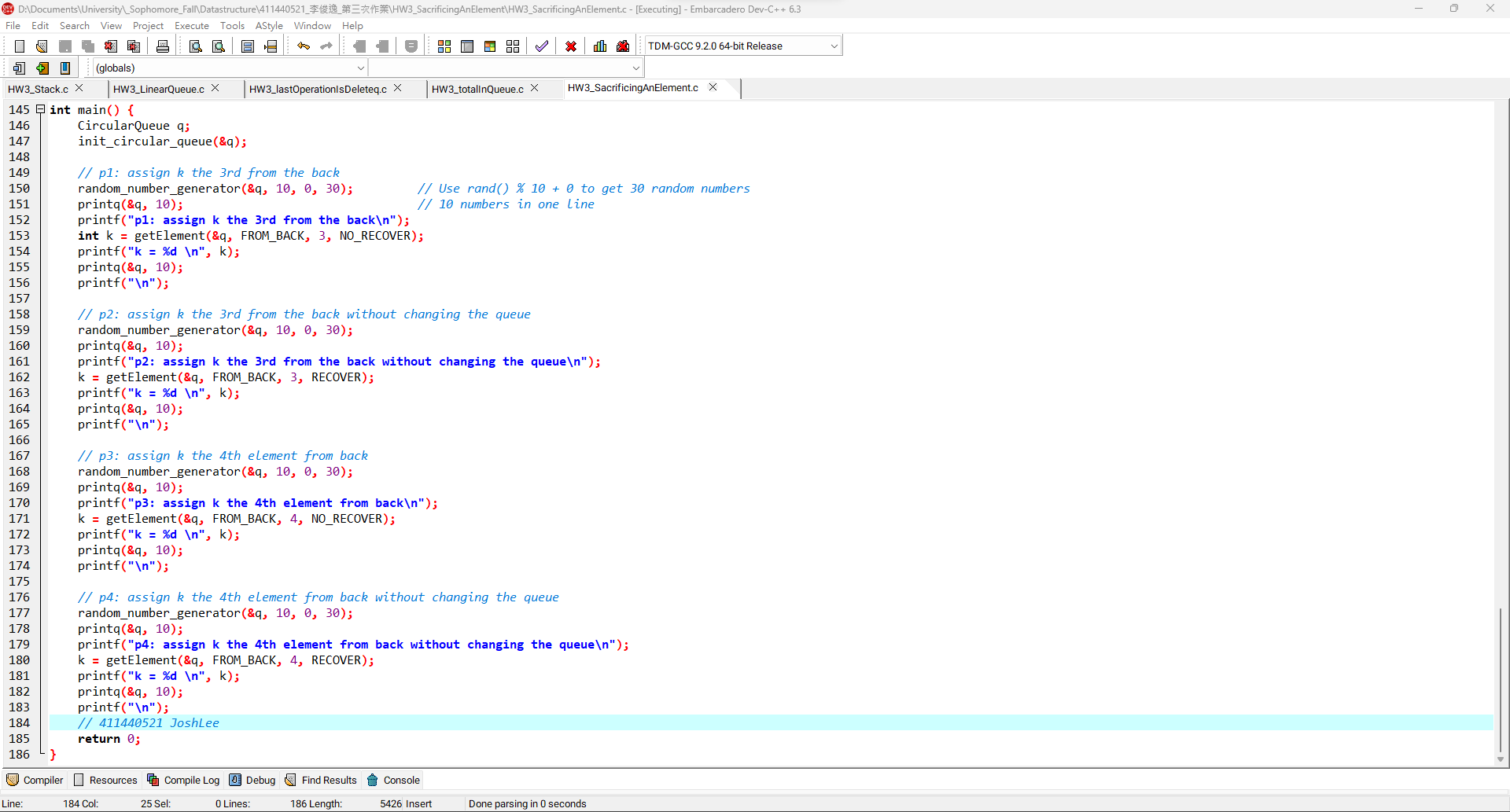
|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Author: 411440521 JoshLee \**  *\* Date: 2023/12/08 \**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define SIZE 100  *// Enumeration for flags*  typedef enum {  FROM\_FRONT,  FROM\_BACK  } Direction;  typedef enum {  RECOVER,  NO\_RECOVER  } RecoveryOption;  typedef struct {  int data[SIZE];  int front;  int rear;  int totalInQueue; *// Total number of elements in the queue*  } CircularQueue;  void init\_circular\_queue(CircularQueue \*q){  q->front = 0;  q->rear = -1;  q->totalInQueue = 0; *// Initially, the queue is empty*  }  *// Circular Queue functions*  int emptyq(CircularQueue \*q) {  return q->totalInQueue == 0;  }  int fullq(CircularQueue \*q) {  return q->totalInQueue == SIZE;  }  *// 411440521 JoshLee*  void addq(CircularQueue \*q, int val) {  if (!fullq(q)) {  q->rear = (q->rear + 1) % SIZE;  q->data[q->rear] = val;  q->totalInQueue++;  } else {  printf("Queue is full!\n");  exit(1);  }  }  int deleteq(CircularQueue \*q) {  if (!emptyq(q)) {  int val = q->data[q->front];  q->front = (q->front + 1) % SIZE;  q->totalInQueue--;  return val;  } else {  printf("Queue is empty!\n");  exit(1);  }  }  *// 411440521 JoshLee*  *// Questions*  void random\_number\_generator(CircularQueue\* q, int range, int offset, int len) {  srand(time(NULL)); *// use srand(time(NULL)) to set seed of random number generator as current time*  int i;  while(!emptyq(q))  deleteq(q);  for(i = 0; i < len; i++)  addq(q, rand() % range + offset); *// generate random number*  }  void recover\_elements(CircularQueue \*ele\_q, CircularQueue \*deleted\_ele\_q) {  while(!emptyq(ele\_q)) { *// deleted\_ele\_q = deleted\_ele\_q + ele\_q*  addq(deleted\_ele\_q, deleteq(ele\_q));  }  while(!emptyq(deleted\_ele\_q)) { *// ele\_q = deleted\_ele\_q*  addq(ele\_q, deleteq(deleted\_ele\_q));  }  }  void printq(CircularQueue\* ele\_q, int ElementsInOneLine) {  CircularQueue temp\_q;  init\_circular\_queue(&temp\_q);  int ElementsInCurrentLine = 0;  while(!emptyq(ele\_q)) {  int deleted\_value = deleteq(ele\_q); addq(&temp\_q, deleted\_value);  printf("%d ", deleted\_value);  if(++ElementsInCurrentLine >= ElementsInOneLine) {  printf("\n");  ElementsInCurrentLine = 0;  }  }  recover\_elements(ele\_q, &temp\_q);  printf(ElementsInCurrentLine ? "\n" : "");  }  *// 411440521 JoshLee*  void reverse\_queue(CircularQueue \*q) {  *// base case*  if (emptyq(q))  return;  *// store front(first element) of queue, and remove front*  int fr = deleteq(q);    *// asking recursion to reverse the leftover queue*  reverse\_queue(q);    *// placing first element at its correct position*  addq(q, fr);  }  int getElement(CircularQueue \*q, Direction direction, int position, RecoveryOption recovery) { *// No Recover*  CircularQueue temp\_q;  init\_circular\_queue(&temp\_q);  int i, deleted\_value;  if(direction == FROM\_FRONT) {  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  } else {  reverse\_queue(q);  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  reverse\_queue(q);  }  return deleted\_value;  }  *// 411440521 JoshLee*  int main() {  CircularQueue q;  init\_circular\_queue(&q);  *// p1: assign k the last element*  random\_number\_generator(&q, 10, 0, 30); *// Use rand() % 10 + 0 to get 30 random numbers*  printq(&q, 10); *// 10 numbers in one line*  printf("p1: assign k the last element\n");  int k = getElement(&q, FROM\_BACK, 1, NO\_RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p2: assign k the last element without changing the queue*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p2: assign k the last element without changing the queue\n");  k = getElement(&q, FROM\_BACK, 1, RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p3: assign k the 2nd element from back*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p3: assign k the 2nd element from back\n");  k = getElement(&q, FROM\_BACK, 2, NO\_RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p4: assign k the 2nd element from back without changing the queue*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p4: assign k the 2nd element from back without changing the queue\n");  k = getElement(&q, FROM\_BACK, 2, RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");  *// 411440521 JoshLee*  return 0;  } |

1

**2**

3

**4**

**5**

**SacrificingAnElement程式碼:**

|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Author: 411440521 JoshLee \**  *\* Date: 2023/12/08 \**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define SIZE 100  *// Enumeration for flags*  typedef enum {  FROM\_FRONT,  FROM\_BACK  } Direction;  typedef enum {  RECOVER,  NO\_RECOVER  } RecoveryOption;  typedef struct {  int data[SIZE];  int front;  int rear;  } CircularQueue;  void init\_circular\_queue(CircularQueue \*q){  q->front = -1;  q->rear = -1;  }  *// Circular Queue functions*  int emptyq(CircularQueue \*q) {  return q->front == -1;  }  int fullq(CircularQueue \*q) {  return (q->rear + 1) % SIZE == q->front;  }  void addq(CircularQueue \*q, int val) {  if(fullq(q)) {  printf("Queue is full!");  exit(1);  } else {  if(emptyq(q))  q->front = 0;  q->rear = (q->rear + 1) % SIZE;  q->data[q->rear] = val;  }  }  int deleteq(CircularQueue \*q) {  if(emptyq(q)) {  printf("Queue is empty!");  exit(1);  } else {  int deletedValue = q->data[q->front];  if(q->front == q->rear) { *// queue is empty*  q->front = -1; *// set front and rear to -1 to symbolize empty state of CircularQueue*  q->rear = -1;  } else {  q->front = ++(q->front) % SIZE; *// use % SIZE to prevent overflow and memory waste*  }  return deletedValue;  }  }  *// 411440521 JoshLee*  *// Questions*  void random\_number\_generator(CircularQueue\* q, int range, int offset, int len) {  srand(time(NULL)); *// use srand(time(NULL)) to set seed of random number generator as current time*  int i;  while(!emptyq(q))  deleteq(q);  for(i = 0; i < len; i++)  addq(q, rand() % range + offset); *// generate random number*  }  void recover\_elements(CircularQueue \*ele\_q, CircularQueue \*deleted\_ele\_q) {  while(!emptyq(ele\_q)) { *// deleted\_ele\_q = deleted\_ele\_q + ele\_q*  addq(deleted\_ele\_q, deleteq(ele\_q));  }  while(!emptyq(deleted\_ele\_q)) { *// ele\_q = deleted\_ele\_q*  addq(ele\_q, deleteq(deleted\_ele\_q));  }  }  void printq(CircularQueue\* ele\_q, int ElementsInOneLine) {  CircularQueue temp\_q;  init\_circular\_queue(&temp\_q);  int ElementsInCurrentLine = 0;  while(!emptyq(ele\_q)) {  int deleted\_value = deleteq(ele\_q); addq(&temp\_q, deleted\_value);  printf("%d ", deleted\_value);  if(++ElementsInCurrentLine >= ElementsInOneLine) {  printf("\n");  ElementsInCurrentLine = 0;  }  }  recover\_elements(ele\_q, &temp\_q);  printf(ElementsInCurrentLine ? "\n" : "");  }  void reverse\_queue(CircularQueue \*q) {  *// base case*  if (emptyq(q))  return;  *// store front(first element) of queue, and remove front*  int fr = deleteq(q);    *// asking recursion to reverse the leftover queue*  reverse\_queue(q);    *// placing first element at its correct position*  addq(q, fr);  }  *// 411440521 JoshLee*  int getElement(CircularQueue \*q, Direction direction, int position, RecoveryOption recovery) { *// No Recover*  CircularQueue temp\_q;  init\_circular\_queue(&temp\_q);  int i, deleted\_value;  if(direction == FROM\_FRONT) {  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  } else {  reverse\_queue(q);  for(i = 0; i < position; i++) {  deleted\_value = deleteq(q);  addq(&temp\_q, deleted\_value);  }  if(recovery == RECOVER) recover\_elements(q, &temp\_q);  reverse\_queue(q);  }  return deleted\_value;  }  *// 411440521 JoshLee*  int main() {  CircularQueue q;  init\_circular\_queue(&q);    *// p1: assign k the 3rd from the back*  random\_number\_generator(&q, 10, 0, 30); *// Use rand() % 10 + 0 to get 30 random numbers*  printq(&q, 10); *// 10 numbers in one line*  printf("p1: assign k the 3rd from the back\n");  int k = getElement(&q, FROM\_BACK, 3, NO\_RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p2: assign k the 3rd from the back without changing the queue*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p2: assign k the 3rd from the back without changing the queue\n");  k = getElement(&q, FROM\_BACK, 3, RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p3: assign k the 4th element from back*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p3: assign k the 4th element from back\n");  k = getElement(&q, FROM\_BACK, 4, NO\_RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");    *// p4: assign k the 4th element from back without changing the queue*  random\_number\_generator(&q, 10, 0, 30);  printq(&q, 10);  printf("p4: assign k the 4th element from back without changing the queue\n");  k = getElement(&q, FROM\_BACK, 4, RECOVER);  printf("k = %d \n", k);  printq(&q, 10);  printf("\n");  *// 411440521 JoshLee*  return 0;  } |