Data structure and algorithm

Assignment W4

Stack and Queue

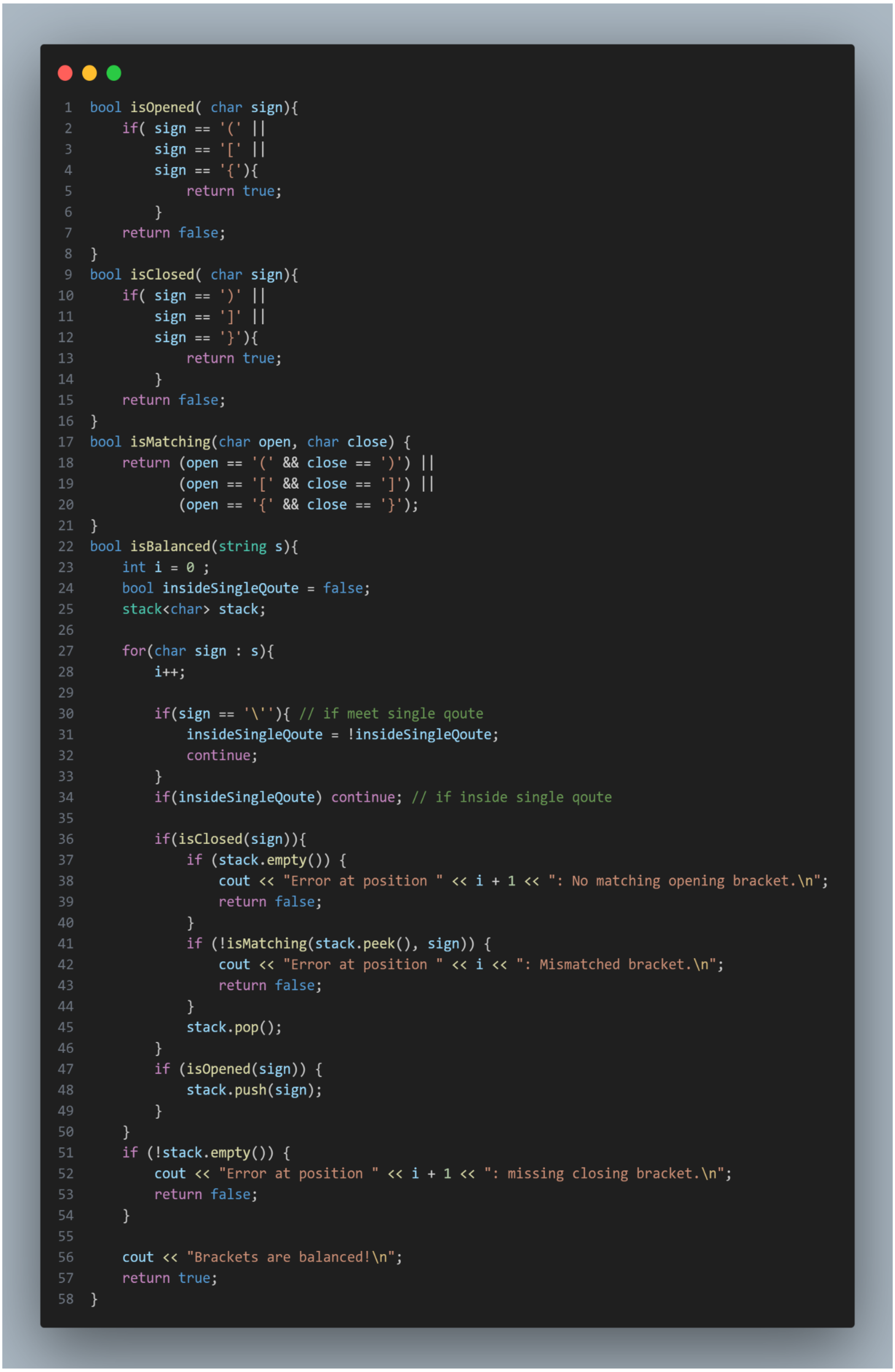
Lecturer: Natt Korat

Group member:

Yeat David	(ID: IDTB110290)
Tiek Chhunhour	(ID: IDTB110259)
Sar Sounamarin	(ID: IDTB110274)
Yam Sosereiwat	(ID: IDTB110267)
Ou Tharatheb	(ID: IDTB110107)

Stack

1. Key code snippets



```cpp
bool isOpened( char sign){
    if( sign == '(' ||
        sign == '[' ||
        sign == '{'){
            return true;
        }
    return false;
}
bool isClosed( char sign){
    if( sign == ')' ||
        sign == ']' ||
        sign == '}'){
            return true;
        }
    return false;
}
bool isMatching(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}');
}
bool isBalanced(string s){
    int i = 0 ;
    bool insideSingleQoute = false;
    stack<char> stack;

    for(char sign : s){
        i++;

        if(sign == '\''){ // if meet single qoute
            insideSingleQoute = !insideSingleQoute;
            continue;
        }
        if(insideSingleQoute) continue; // if inside single qoute

        if(isClosed(sign)){
            if (stack.empty()) {
                cout << "Error at position " << i + 1 << ": No matching opening bracket.\n";
                return false;
            }
            if (!isMatching(stack.peek(), sign)) {
                cout << "Error at position " << i << ": Mismatched bracket.\n";
                return false;
            }
            stack.pop();
        }
        if (isOpened(sign)) {
            stack.push(sign);
        }
    }
    if (!stack.empty()) {
        cout << "Error at position " << i + 1 << ": missing closing bracket.\n";
        return false;
    }

    cout << "Brackets are balanced!\n";
    return true;
}
```

- The stack property is LIFO: that suits what we need because the last open curly bracket must be the first closed.
- Tracking the position of error: Check all elements one by one. If it has any errors, it will track and continue checking.
- Use three other helping functions (isOpen, isClosed, isMatching): This way the code can run smoothly and readability.

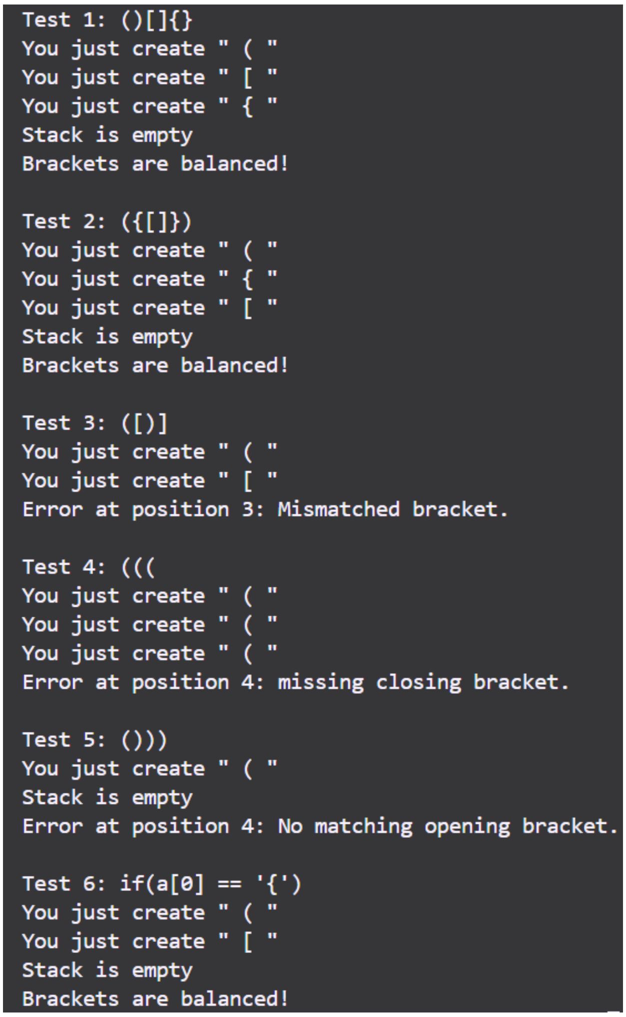
➢ Edge case handle:

- Empty string: it will return true or balance
- Only opening brackets: detect whether there are any missing closing brackets or not
- Only closing brackets: detect whether the opening bracket is matched to closing bracket or not
- Single bracket: detect unbalanced
- Mixed content: Ignores non-bracket characters correctly
- Mismatched types: Detects when bracket type don't match
- Deep Nesting: Handle multiple levels of nested brackets

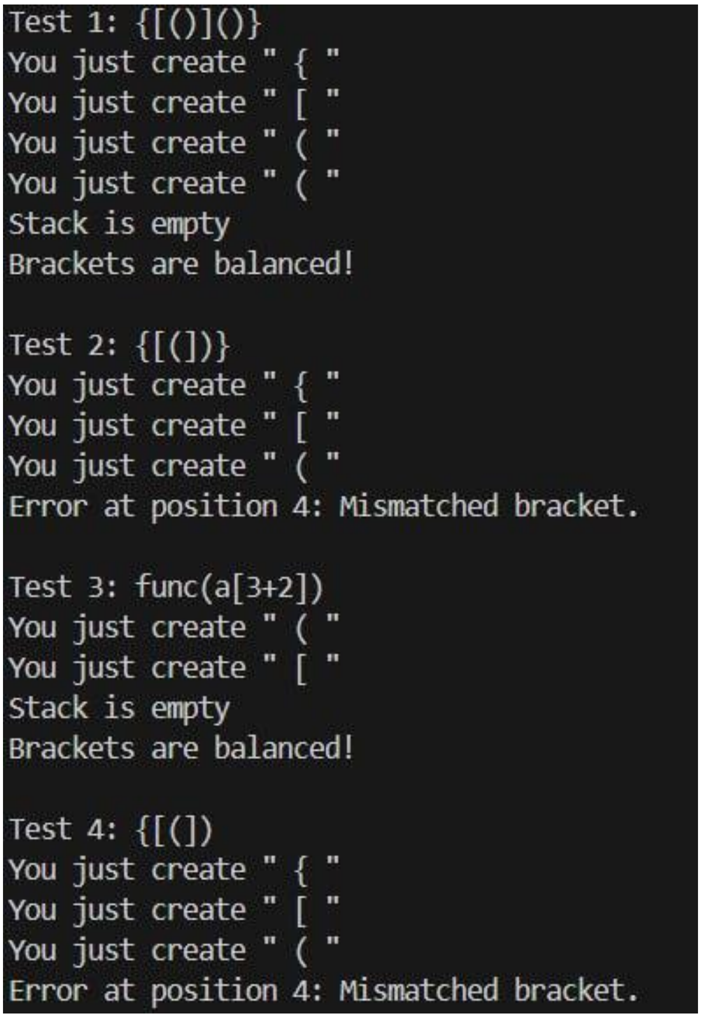
➢ Operation: This operation is enough because:

- Push: Store opening bracket.
- Pop: Remove matched pairs.
- Peek: Check what needs to be closed next.
- Empty: Verify that all brackets are matched.

3. Result (screenshot):



```
Test 1: ()[]{}
You just create " ( "
You just create " [ "
You just create " { "
Stack is empty
Brackets are balanced!

Test 2: ({[]})
You just create " ( "
You just create " { "
You just create " [ "
Stack is empty
Brackets are balanced!

Test 3: ([)]
You just create " ( "
You just create " [ "
Error at position 3: Mismatched bracket.

Test 4: (((
You just create " ( "
You just create " ( "
You just create " ( "
Error at position 4: missing closing bracket.

Test 5: ()))
You just create " ( "
Stack is empty
Error at position 4: No matching opening bracket.

Test 6: if(a[0] == '{')
You just create " ( "
You just create " [ "
Stack is empty
Brackets are balanced!
```



```
Test 1: {[()]()}
You just create " { "
You just create " [ "
You just create " ( "
You just create " ( "
Stack is empty
Brackets are balanced!

Test 2: {[(])}
You just create " { "
You just create " [ "
You just create " ( "
Error at position 4: Mismatched bracket.

Test 3: func(a[3+2])
You just create " ( "
You just create " [ "
Stack is empty
Brackets are balanced!

Test 4: {[(])
You just create " { "
You just create " [ "
You just create " ( "
Error at position 4: Mismatched bracket.
```

Queue

Scenario: Coffee shop

- Design: We use arrays instead of linked lists. It's because the Caffee Shop has limited space for customers to wait. Access speed faster than linked list only enqueue /dequeue needed, no random insert/delete. And it takes lower memory allocation than the Linked list. It also takes less time to access, which is better than the linked list.
  - Diagram:
    In our Coffee Shop have fixed size 8 people for waiting:

    Index: 0 1 2 3 4 5 6 7
    Queue: [-][-][-][-][-][-][-][-]
    Front, rear (empty queue)

    1. Customer Arrives (Enqueue)
       Index: 0 1 2 3 4 5 6 7

```
Queue:  [ A ][ B ][ C ][ D ][ - ][ - ][ - ][ - ]
          ↑               ↑
        front            rear
```

2. Customer Served (Dequeue)

```
Index:      0   1    2    3   4   5   6   7
Queue:  [ - ][ B ][ C ][ D ][ - ][ - ][ - ][ - ]
              ↑          ↑
            front      rear
```

3. New Customer Join (Enqueue)

```
Index:      0   1    2    3    4    5   6   7
Queue:  [ - ][ B ][ C ][ D ][ E ][ F ][ - ][ - ]
              ↑                    ↑
            front                 rear
```

4. When Queue is Full

```
Index:      0    1    2    3     4     5   6   7
Queue:  [ A ][ B ][ C ][ D ][ E ][ F ][ G ][ H ]
          ↑                                   ↑
        front                                rear
```

- Methods

Methods’ name	Behaviors
Is full	To check in Queue list is full or not
Is empty	To check in Queue list is empty or not
enqueue	To store data of the order in the queue
dequeue	To remove the order from queue and serve next order
peek	To view the first order in the queue
Size	To return current size in queue list
capacity	To return Max size of queue list
display	To display queue list

- Edge cases handled:

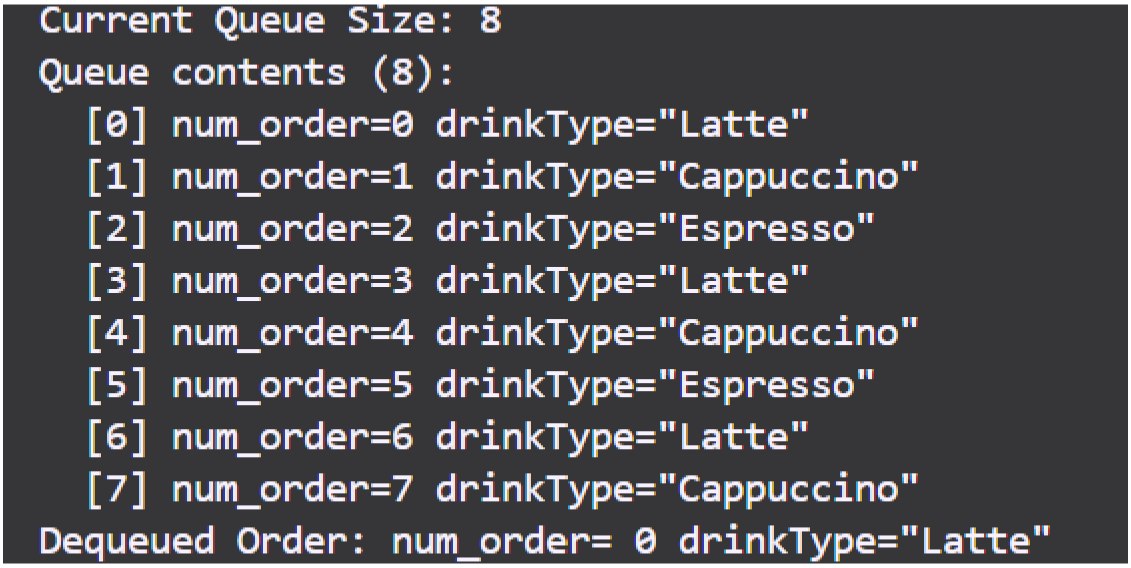
1. Queue Empty: when trying to dequeue() or peek() but no customers are in line first we check if (front == rear)

2. Queue Full: when trying to enqueue() but in lines are full we handle location by check if(rear == Max_Size - 1)
3. Memory safety: Bound checks before enqueue/dequeue

- Complexity

Operation	Time complexity	Spaces cost
Is full/ Is empty	O(1)	O(1)
Enqueue	O(1)	O(1)
Dequeue	O(1)	O(1)
Peek	O(1)	O(1)

- Evidence of correctness



```
Current Queue Size: 8
Queue contents (8):
  [0] num_order=0 drinkType="Latte"
  [1] num_order=1 drinkType="Cappuccino"
  [2] num_order=2 drinkType="Espresso"
  [3] num_order=3 drinkType="Latte"
  [4] num_order=4 drinkType="Cappuccino"
  [5] num_order=5 drinkType="Espresso"
  [6] num_order=6 drinkType="Latte"
  [7] num_order=7 drinkType="Cappuccino"
Dequeued Order: num_order= 0 drinkType="Latte"
```