C++

Template <class T, class Container = deque<T> >

Class stack;

FIFO

Add()/enqueue()

Void enQueue(int value);

Remove()/deque()

Void deque();

LIFO

Add(x)/push(x)

Void push(const value\_type&val);

Remove()/pop()

Value\_type& top();

Void pop();

Priority Queue

deleteMin()

ItemType deleteMin();

DEQUE

addFirst()

dequename.push\_front(value);

removeFirst()

dequename.pop\_front();

addLast()

dequename.push\_back(value);

removeLast()

dequename.pop\_back();

List

Size()

list\_name.size();

get(i)

bool search(Node \*head, int x)

set(i,x)

void replace (ForwardIteraor first, ForwardIterator last, const T& old\_value, const T& new\_value)

add(i,x)

list\_name.assign(count, value);

remove(i)

list\_name.remove(val);

USET

Size()

Unordered\_set\_name.size()

Add()

Unordered\_set\_name.insert(value)

Remove()

Unordered\_set\_name.erase(element)

Java

FIFO

Add()/enqueue()

Public void enqueue(int item)

Remove()/deque()

Public void deque(int item)

LIFO

Add(x)/push(x)

STACK.push(*E* element)

Remove()/pop()

STACK.pop()

Priority QUEUE

deleteMin()

Priority\_Queue.remove(Object O)

DEQUE

addFirst()

void addFirst(E e)

removeFirst()

Array\_Deque.removeFirst()

addLast()

Boolean addLast(E e)

removeLast()

Array\_Deque.removeLast()

List

Size()

Public int size()

Get(i)

E get (int index)

Set(i,x)

Public void add(int index, E element)

Add(i,x)

Public void add(int index, E element)

Remove()

E remove(int index)

USET

Size()

Int size()

Add()

Boolean add(E element)

Remove()

Boolean remove(Object O)

Find()

Boolean contains(Object element)

SSET

Find()- finds x in a sorted set

E first()

Python

FIFO

Add()/enqueue()

Queue.append(element)

remove()/deque()

queue.pop(element)

Priority Queue

deleteMin()

DEQUE

addFirst()

de.append(element)

removeFirst()

de.pop(element)

addLast()

de.appendLeft(element)

removeLast()

de.popLeft(element)

List

size()

len(list\_name)

get()

list\_name.index(element, start, end)

set(i,x)

set(iterable)

add(i,x)

list\_name.add(obj)

remove(x)

list\_name.remove(obj)

USET

size()

len(set\_name)

add()

set\_name.add(item)

remove()

set\_name.remove(item)

find()

str.find(sub,start,end)

SSET

find()- finds x in a sorted set

3. First, create and initialize a stack. Get the input and confirm that +1 is a push and -1 is a pop. Repeat this for each element in the sequence. Using the push function, if the element is +1, push it into the stack and if the element is -1, pop the element. If there is nothing in a stack and you try popping from an empty stack, this will not result in a dyck word. If there is a sum of 0 or a positive number, it’s a dyck word.

4. To determine if a string is matched using a stack, you would start by declaring a stack. For every element in the given string, check what element it is. If any element is ‘{‘, ‘[‘, or ‘(‘, which is any opening bracket, append that element into the declared stack. If every element is checked in the string, return False because this means that there are no closing brackets which means the string is not a matched string. If any element is ‘}’, ‘]’, or ‘)’, any closing bracket, pop one element from the stack and check if that element is a matching opening bracket with the closing bracket. If the popped element is a closing bracket with the same type, then we continue, but if it is not a closing bracket, the string is not a matched string After checking for all the elements in the string, if there is a still an opening bracket in the stack, it is not a matched string because if there is the same amount of closing brackets with opening brackets, then all the opening brackets from our stack should be popped.

5. Starting with a FIFO Queue, you would push each element, one by one, into Stack s. The first element that was in the Queue will now be at the bottom of the stack s. Since the Queue q is FIFO (First In First Out), when popping each element off of Stack s back into the Queue, the top of the stack, which was the bottom of the Queue, will be popped off first and added to the queue. The first elements popped off the stack will be the first elements in the new Queue which reverses the order from the original.

6. A Bag gives you the option to add repeated elements into a set. First, initialize a set and the class Bag. Then, define add to determine whether an element that equals “x” is in the set. Then add an if else statement to see if the element exists. If it does exist, you would add one more, but if not, you would add. Next, define remove where the element “x” is decreased by 1 if it exists, but if it doesn’t exist, nothing happens. Defining the find function checks if the element “x” exists in the set and if it does, “x” is returned. If it doesn’t, then nothing is returned. The findAll function checks if “x” exists in the set and returns all elements that equal to “x” and returns nothing if no elements equal to “x”.