DATE : ___/___/_____

Data Structures

Submitted To:

Dr. Anayat ur Rehman

Submitted By:

Muhammad Rizwan Shafiq

SP24-BCS-069

BCS-3B

Assignment # 02

DATE: __/__/____

(Question 01)

(i)

"Bank Simulation"

Suitable: No

Reason: Bank simulation usually serve the customers in order they arrive (First-In-First-Out) but Stack is based on Last-In-First-Out. So, its better to use Queue.

(ii)

"Address Book"

Suitable: No

Reason: It requires random access or sorted order but a stack's LIFO nature limits the access to only top element A list or Array would be better here.

(iii)

"Recieve data & process in Reverse"

Suitable: Yes

Reason: It matches the LIFO property of Stack perfectly Data can be pushed as recieved and then popped off to process in reverse.

(iv)

"Word Processor to have Pf key"

Suitable: Yes

Reason: Its similar to "undo". Commands can be pushed to stack and pressing Pf key pops the top command

DATE: __/__/____

that was the previous one.

(v)

"Program to evaluate Arithematic Expression"

Suitable: Yes

Reason: Stacks are commonly used to evaluate the expressions i.e Postfix notations.

(vi)

"Dictionary"

Suitable: No

Reason: It needs fast lookup and a sorted order. A stacks LIFO restriction makes retrival inssufficent unless accessing most recently added word.

(vii)

"Data structure to keep track of return addresses"

Suitable: Yes

Reason: Return addresses are pushed when a function is called and popped when it returns, perfectly matching the LIFO nature. It also describes the call Stack.

(viii)

"Program to keep track of patients"

Suitable: No

Reason: First-come-first-served is FIFO not LIFO, So a queue would be more appropriate here as stack would assign the most recently arrived patient first.

DATE: __/__/____

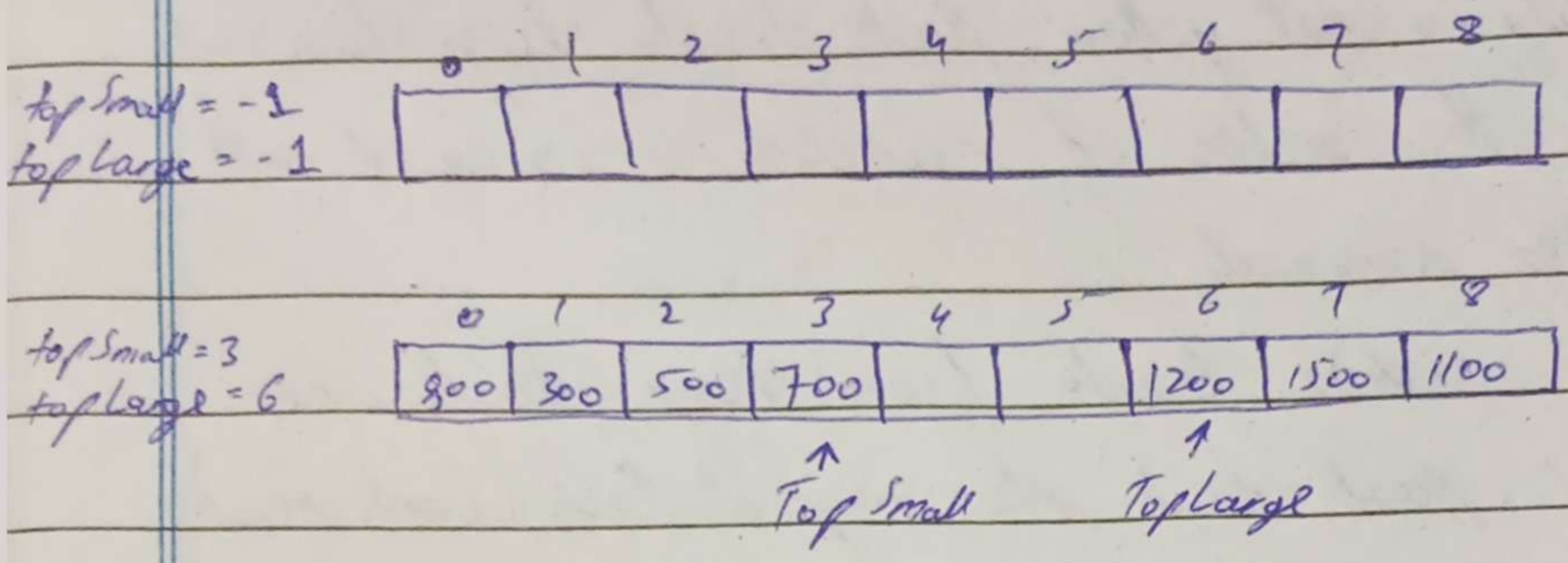
(Question 02)

a) Diagram:

⇒ Two top pointers

⇒ TopSmall (for values <=1000) starts from 0th index

⇒ Tophauge (for values >1000) starts from size-1 index



b) Defination:

```
const int SIZE = 200;
class DoubleStack {
    int data[SIZE];
    int topSmall;
    int topLarge;

    DoubleStack() {
        topSmall = -1; topLarge = SIZE;
    }
```

c) Algo for push:

```
void push(int item) {
    if (topSmall + 1 >= topLarge) {
        cout << "Stack is Full" << endl;
        return;
    }
    if (item <= 1000) {
        data[topSmall++] = item;
    } else {
        data[topLarge--] = item;
    }
```

DATE: __/__/____

(Question 03)

"Remove Yellow Candies"

① Use two stacks, originalStack and a dynamic tempStack.
② Take out candies one by one from original stack.
and if candy is not yellow push it into tempStack.
③ In this way the order of candies is reversed with
yellow candies removed.
④ Now pop candies back from temp stack one
by one and push back into originalStack (container).

```
void filterYellowCandies(Stack & originalStack.){
    Stack tempStack;

    while(!originalStack.isEmpty()){
        string candy = originalStack.pop();
        if(candy ≠ "yellow"){
            tempStack.push(candy);
        }
    } // Reversed orde of candies with yellow ones removed

    while(!tempStack.isEmpty()){
        string candy = tempStack.pop();
        originalStack.push(candy);
    } // Transfer back non-yellow candies.
}
```

DATE: __/__/____

Question 04

Infix to Postfix

1. $(A+B)^*(C-D)$

Itration	Exp[char]	Stack	Postfix
1	(	(	
2	A	(	A
3	+	(+	A
4	B	(+	AB
5	)		AB+
6	*	*	AB+
7	(	*(	AB+
8	C	*(	AB+C
9	-	*(-	AB+C
10	)	*	AB+C-
			AB+C-*

Answer: AB+C-*

2. $A\hat{}B^*C-D+E/F$

Itration	Exp[char]	Stack	Postfix
1	A		A
2	^	^	A
3	B	^	AB
4	*	*	AB^
5	C	*	AB^C
6	-	-	AB^C*
7	D	-	AB^C*D

DATE: ___/___/_____

8	+	+	AB^C*D-
9	E	+	AB^C*D-E
10	/	+/	AB^C*D-E
11	F	+/	AB^C*D-EF
			AB^C*D-EF/+

Answer: AB^C*D-EF/+

3. A / (B+C * D-E)

Itration	Exp[char]	Stack	Postfix
1	A		A
2	/	/	A
3	(	/(	A
4	B	/(	AB
5	+	/(+	AB
6	C	/(+	ABC
7	*	/(+*	ABC
8	D	/(+*	ABCD
9	-	/(-	ABCD*+
10	E	/(-	ABCD*+E
11	)	/	ABCD*+E-
			ABCD*+E-/

Answer: ABCD*+E-/

4. A - B * C + D / E

DATE: __/__/____

Itration	Exp[char]	Stack	Postfix
1	A		A
2	-	-	
3	B	-	AB
4	*	-*	AB
5	C	-*	ABC
6	+	+	ABC*-
7	D	+	ABC*-D
8	/	+/	ABC*-D
9	E	+/	ABC*-DE
			ABC*-DE/+

Answer: ABC*-DE/+

5. (A+B)^2 - (C-D)/2

Itration	Exp[char]	Stack	Postfix
1	(	(	
2	A	(	A
3	+	(+	A
4	B	(+	AB
5	)		AB+
6	^	^	AB+
7	2	^	AB+2
8	-	-	AB+2^
9	(	-(	AB+2^
10	C	-(	AB+2^C
11	-	-(-	AB+2^C
12	D	-(-	AB+2^CD
13	)	-	AB+2^CD-
14	/	-/	AB+2^CD-
15	2	-/	AB+2^CD-2

Answer: AB+2^CD-2/-

DATE : ___ /___ /_____

(Question 05)

Evaluate Postfix

A = 12 , B = 3 , C = 7 , D = 4 , E = 2 , F = 5

1. AB + CD - *

Itration	Vallue	Stack
1	12	12
2	3	12, 3
3	+	(12+3)=15
4	7	15, 7
5	4	15, 7, 4
6	-	15, (7-4)=3
7	*	(15*3) = 45

2. AB^C*D-EF/+

Itration	Value	Stack
1	12	12
2	3	12, 3
3	^	(12^3)=1728
4	7	1728, 7
5	*	(1728*7)=12,096
6	4	12,096, 4
7	-	(12096-4)=12092
8	2	12092, 2
9	5	12092, 2, 5
10	/	12092, (2/5)=0.4
11	+	(12092+0.4)=12092.4

Answer: 12092.4

DATE: __/__/____

3. ABCD * + E - /

Itration	Value	Stack
1	12	12
2	3	12,3
3	7	12,3,7
4	4	12,3,7,4
5	*	12,3,(7*4)=12,3,28
6	+	12,(3+28)=31
7	2	12,31,2
8	-	12,(31-2)=29
9	/	12/29 = 0.414

4. ABC * - DE / +

Itration	Value	Stack
1	12	12
2	3	12,3
3	7	12,3,7
4	*	12,(3*7)=21
5	-	(12-21)=-9
6	4	-9,4
7	2	-9,4,2
8	/	-9,(4/2)=2
9	+	(-9+2)= -7

5. AB + 2 ^ CD - 2 / -

Itration	Value	Stack
1	12	12
2	3	12,3

DATE ___/___/_____

Itration	Value	Stack
3	+	(12+3) = 15
4	2	15, 2
5	^	(15^2) = 225
6	7	225, 7
7	4	225, 7, 4
8	-	225, (7-4) = 3
9	2	225, 3, 2
10	/	225, (3/2) = 1.5
11	-	(225-1.5) = 223.5

Question 06)

i) Algorithm to convert infix to Prefix:

```
String infixToPrefix(string infix){
    Stack operatorStack;
    string prefix = "";

    // Step 01: Reverse infix and swap parenthesis
    string reversed = "";
    for(int i = infix.length()-1; i>=0; i--){
        if(infix[i] = '(')  reversed += ')';
        else if(infix[i] = ')') reversed += '(';
        else reversed += infix[i];
    }
```

DATE: __/__/____

```
// Step 02: Process reversed infix
for (char c : reversed) {
    if (isalnum(c)) {
        prefix += c; // Add operand to prefix
    } else if (c == '(') {
        operatorStack.push(c); // push opening (
    } else if (c == ')') { // for closing )
        while (!operatorStack.isEmpty() && operatorStack.peek() != '(') {
            char op = operatorStack.pop(); // pop and add to
                                           prefix until )
            prefix += op;
        }
        if (!operatorStack.isEmpty()) {
            operatorStack.pop(); // discard '('
        }
    } else { // for operators
        while (!operatorStack.isEmpty() &&
               operatorStack.peek() != '(' && // loop before '(' is reached in
                                                 Stack
               precedence(operatorStack.peek()) >= precedence(c))
            // precedence of operator in Stack is > precedence of current c
            char op = operatorStack.pop(); // pop and add to prefix
            prefix += op;
        }
        operatorStack.push(c);
    }
}

// Step 3: pop remaining operators and add to Prefix
while (!operatorStack.isEmpty()) {
    char op = operatorStack.pop();
    if (op != ')' && op != '(') { // Skip parenthesis if any.
        prefix += op;
    }
```

DATE: __/__/____

Step 04: Reverse the result to have final Answer.

```
string result = "";
for (int i = prefix.length-1 , i >=0 , i-- ){}
        result += prefix[i];
    }

    return result;
}
```

```
// Helper function to get operator Precedence
int precedence (char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-' return 1;
    return 0;
}
```

ii) Algorithm to evaluate prefix Expression

```
double evalPrefix (string prefix) {
    Stack st;

// Process from right to left.
for (int i = prefix.length()-1; i>=0 ; i--){
    char t = prefix[i]; // one char at a time
    if (isdigit(t)) { // assuming single digit numbers
        st.push(t - '0'); // convert char to double.
} else if (t == '+' || t == '-' || t == '*' || t == '/' # t == '^
    double operand1 = st.pop();
    double operand2 = st.pop();
    double result;
```

DATE: __/__/____

```
switch (t) {
        case '+': result = operand1 + operand2; break;
        case '-': result = operand1 - operand2; break;
        case '*': result = operand1 * operand2; break;
        case '/': result = operand1 / operand2; break;
        case '^': result = pow(operand1, operand2); break;
        default: result = 0;
    }
    st.push(result);
}
return st.pop();
}
```

(Question 07)

i) Infix to Prefix = A^B*C-D+E/F/(G+G)

Reverse = (G+G)/F/E+D-C*B^A

Iteration	Exp[char]	Stack	Result
1	(	(	
2	G	(	G
3	+	(+	G
4	G	(+	GG
5	)		GG+
6	/	/	GG+
7	F	/	GG+F
8	/	/	GG+F/
9	E	/	GG+F/E
10	+	+	GG+F/E/

DATE: __/__/____

11	D	+	GG+F/E/D
12	-	-	GG+F/E/D+
13	C	-	GG+F/E/D+C
14	*	-*	GG+F/E/D+C
15	B	-*	GG+F/E/D+CB
16	^	-*^	GG+F/E/D+CB
17	A	-*^	GG+F/E/D+CBA
			GG+F/E/D+CBA^*-

Reversing = -*^ABC+D/E/F+GG

ii, Evaluate Prefix:

+A*B+CD A=2, B=3, C=4, D=5

Itration	Exp[char]	value	Stack
1	D	5	5
2	C	4	5, 4
3	+	+	(5+4)=11
4	B	3	11, 3
5	*	*	(11*3)=33
6	A	2	33, 2
7	+	+	(33+2)=35

Question 08

a) Iterative Algo:

```
int a, b;
cin >> a ; cin >> b;
int result = a;
for (int i = 1; i ≤ b - 1 ; i++) {
    result += a;
}
cout << result;
```

DATE: __/__/____

b) Recursive: $a * b = a(b+1) - a$;

⇒ No, it is not possible because this defination increases b leading to infinite recursion with no natural base case

(Question 09)

Queue Implementation

a) Employee Record (Recently hired paid off first)
Not suitable as Queue is FIFO, Stack will be used.

b) Patients
Yes, Queue is AFO (First come first served)

c) Backtrack maze to earlier position.
Not suitable, Stack will be suitable due to LIFO, which tracks most recent and is useful for 'undo'.

d) Parts Inventory
Not suitable, Queue only manages sequence wise. Random access is required i.e. Array or List is better.

e) Operating System Request in order.
Yes, Queue manages data in order upon requested.

f) Grocery check customer wait time
Yes, have to serve the customer in orderly manner. (FIFO).

DATE : __/__/____

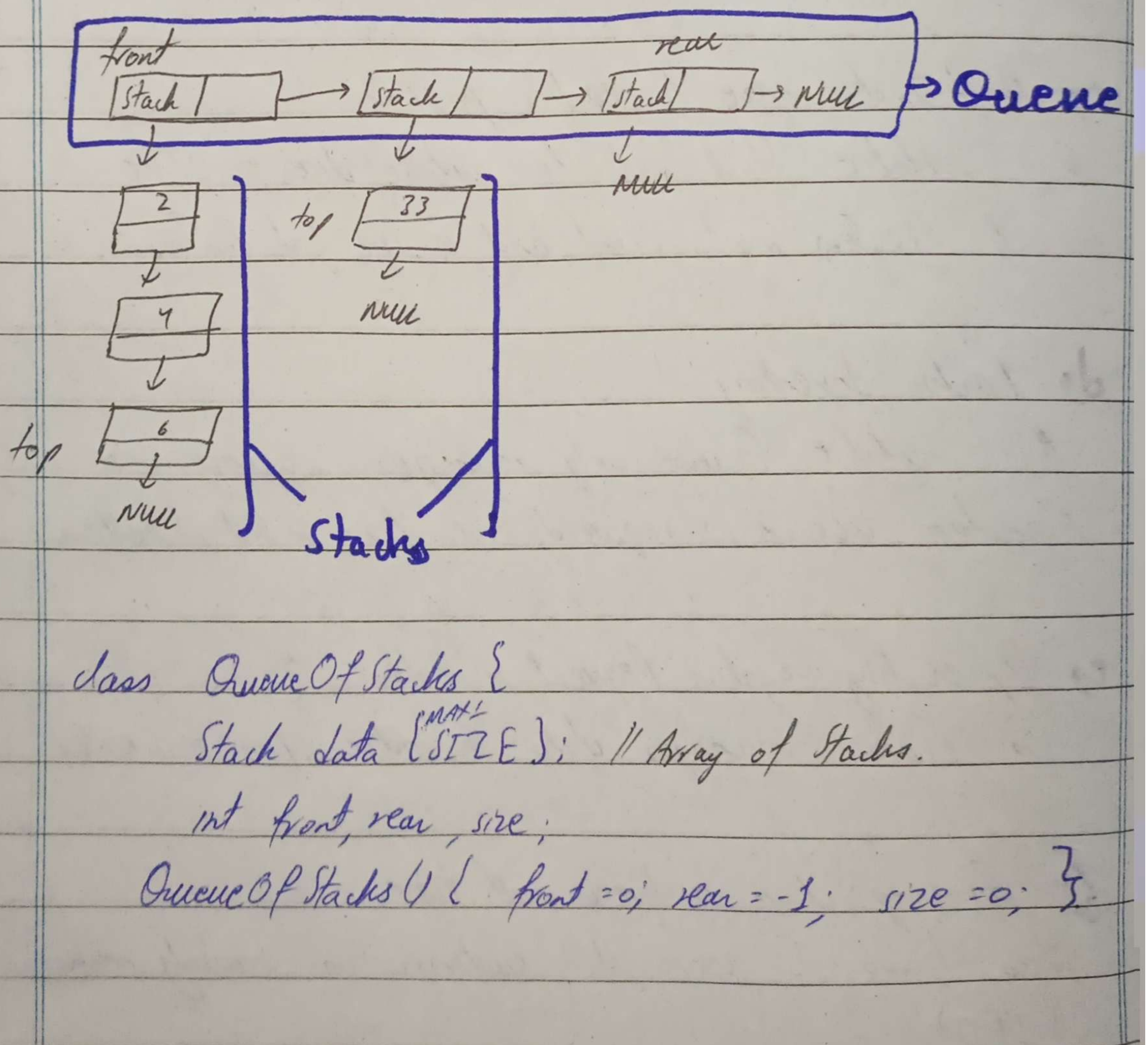
g) Dictionary used to spelling checker
Not suitable, Queue only reads data in one order-
very slow

h) Bakery customers severed in order.
Yes, Queue is FIFO.

i) Gamblers taking numbers in lottery
No, Since they have to pick random numbers, so
queue is unsuitable for this

Question 10.

i) Queue Of Stacks.

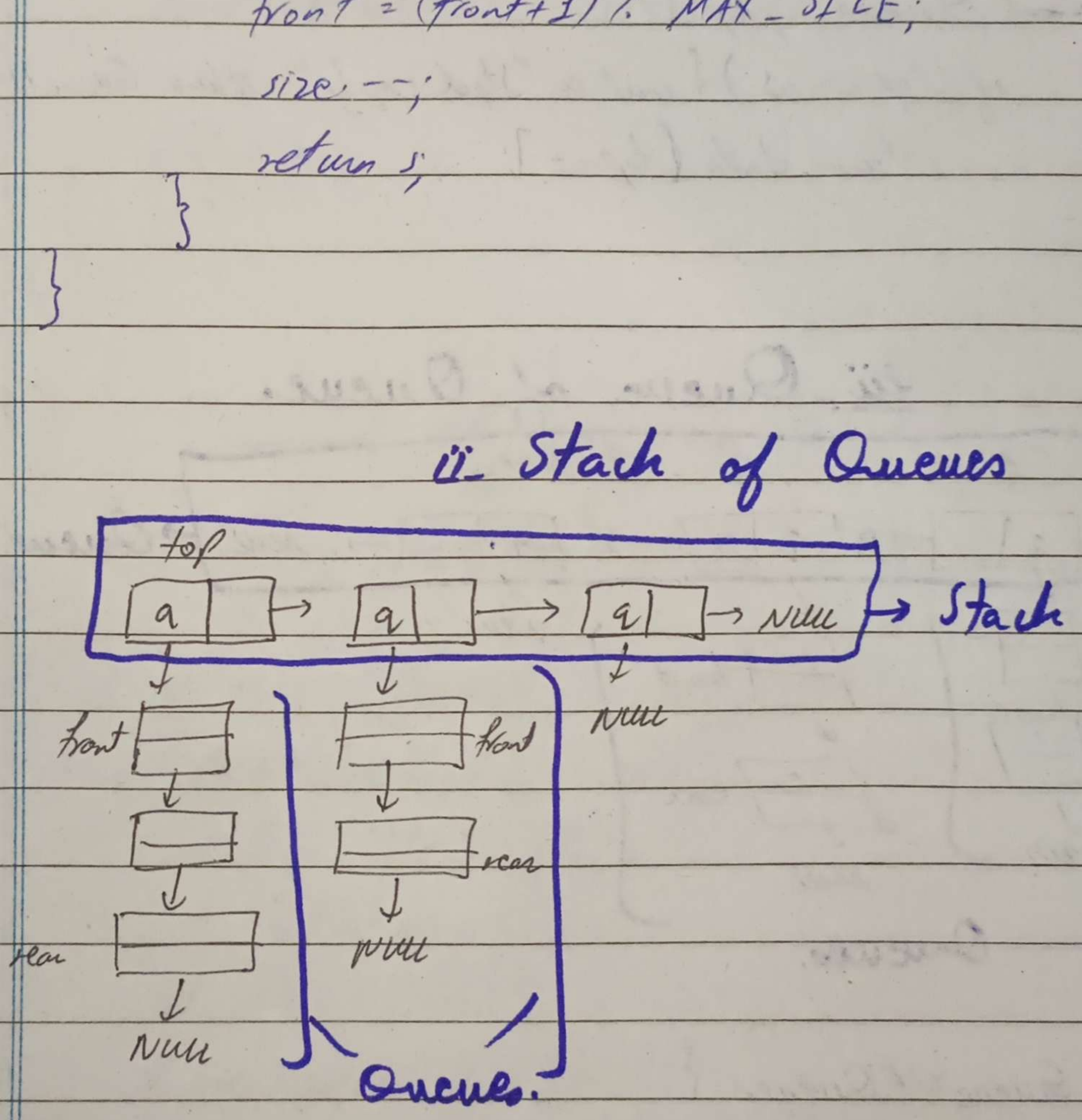



```
class QueueOfStacks {
    Stack data [MAX_SIZE]; // Array of Stacks.
    int front, rear, size;
    QueueOfStacks() { front = 0; rear = -1; size = 0; }
```

DATE: __/__/____

```
void enqueue (Stack s){
        if (size == MAX_SIZE) { cout << "Queue Full" << endl; return; }
        rear = (rear + 1) % MAXSIZE;
        data [rear ] = s;
        size ++;
}

Stack dequeue (){
        if (size == 0) { cout << "Que Empty"; return Stack(); }
        Stack s = data [front ];
        front = (front+1) % MAX_SIZE;
        size. --;
        return s;
        }
}
```

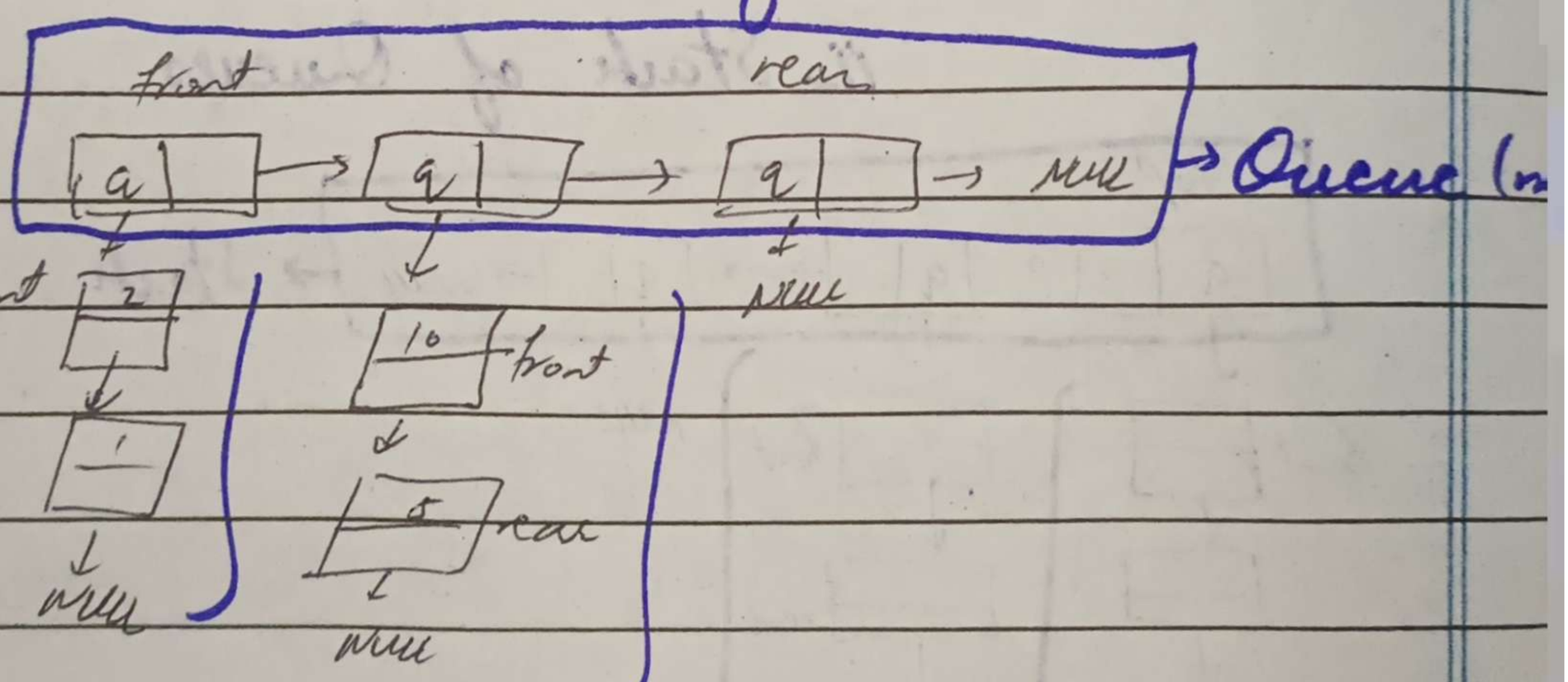
ii. Stack of Queues



DATE __/__/____

```
class StackOfQueues{
    Queue data[MAX_SIZE];
    int top;
    StackOfQueues() { top = -1; }

    void push(Queue q) {
        if (top == MAX_SIZE-1) { cout << "Stack full!"; return;
            data[++top] = q;
        }

    Queue pop() {
        if (top == -1) { cout << "Stack empty"; return Queue(); }
            return data[top--];
    }
```

iii. Queue of Queues



Queues.

```
class QueueOfQueues {
    Queue data[MAX_SIZE];
    int front, rear, size;
    QueueOfQueues() { front = 0; rear = -1; size = 0; }
```

DATE: __/__/____

```
void enqueue (Queue q) {
    if (size == MAX_SIZE) { cout << "Queue full"; return; }
    rear = (rear + 1) % MAX_SIZE;
    data [rear] = q;
    size ++;
}
```

```
Queue dequeue () {
    if (size == 0) { cout << "Queue Empty"; return Queue(); }
    Queue q = data [front];
    front = (front + 1) % MAX_SIZE;
    size --;
    return q;
  }
}
```

Question 11

Two Queues in one array.

```
const int MAX_SIZE = 10;
class DoubleQueue {
    int data [MAX_SIZE];
    int front1, rear1;
    int front2, rear2;
    DoubleQueue () {
        front1 = 0; rear1 = -1; // from left
        front2 = MAX_SIZE - 1; rear2 = MAX_SIZE; // from Right
    }
```

DATE : __/__/____

```
bool isFull() {
    return (rear1 > rear2 || rear2 < rear1) && rear1 != -1
        && rear2 != MAX_SIZE;
}
```

```
bool isQueue1Empty() { return rear1 < front1 }
bool isQueue2Empty() { return rear2 > front2 }
```

```
void enqueueQue1 (int item) {
    if (isFull()) { return; }
    data [++rear1] = item;
}
```

• same for enqueQ

→ data [--rear2] = item;

```
int dequeueQue1() {
    if (isQueue1Empty()) { return; }
    int item = data [front1++];
    return item;
}
```

« same for Que 2

→ int item = data [front--];

(Question 12)
Reverse Circular Queue.

1- Declare dynamic stacks in program.

```
Stack s;
```

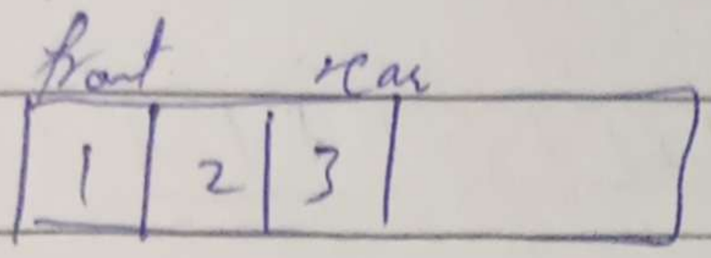
2- Deque elements from Queue and push them into stack.

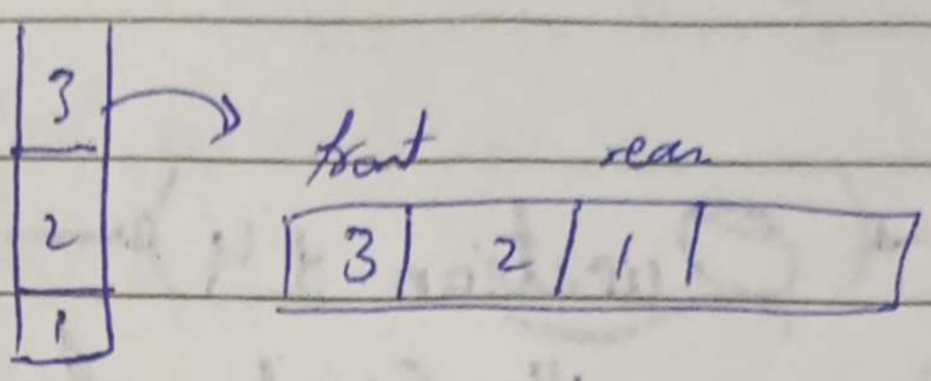
```
while (!(isEmpty()) {
    s.push (dequeue());
}
```

DATE: __/__/____

3- Now pop out elements from stack, they will be in reverse order and push then back in que

```
while (stop != NULL) {
        enque (s.pop());
}
```



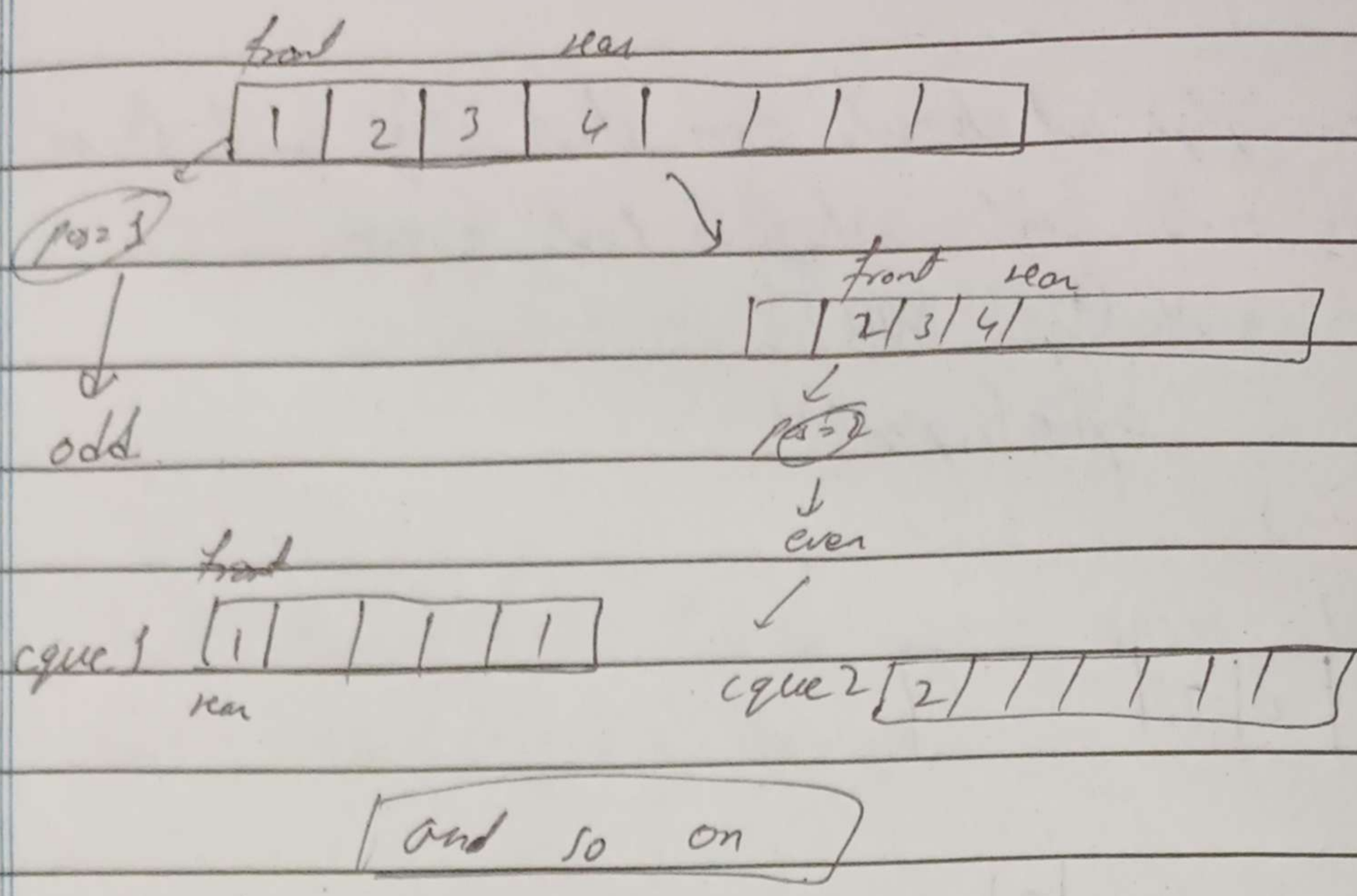


Question 13)

Split Circular Queue

```
void splitCircularQue (Queue & original, Queue& que1, Queue & que2){
    if (original.isEmpty()){
        cout << "Original Queue Empty";
        return;
    }

    int position = 1;
    int index = original.getFront();
    int totalelements = original.getSize();

    for (int i=0; i<totalelements; i++) {
        ~~int item = original.dequeue()~~
        int item = original.getData(index);
        if (position % 2 == 1) {
            que1.enqueue(item);
        }else{
            que2.enqueue(item);
```

DATE __/__/____



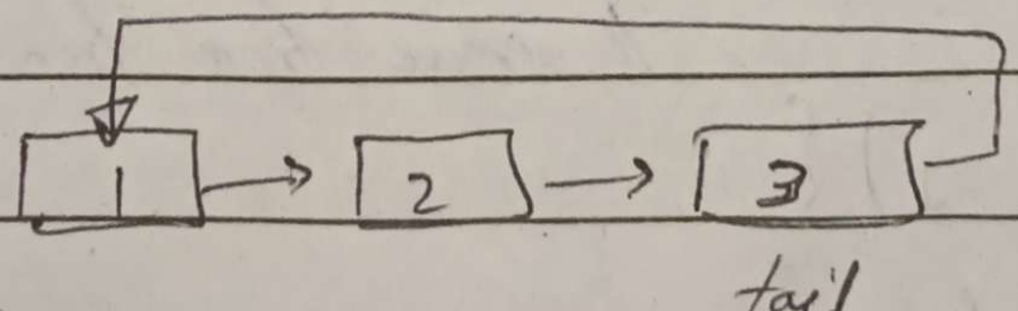
(Question 14)

Queue with Circular linked list.

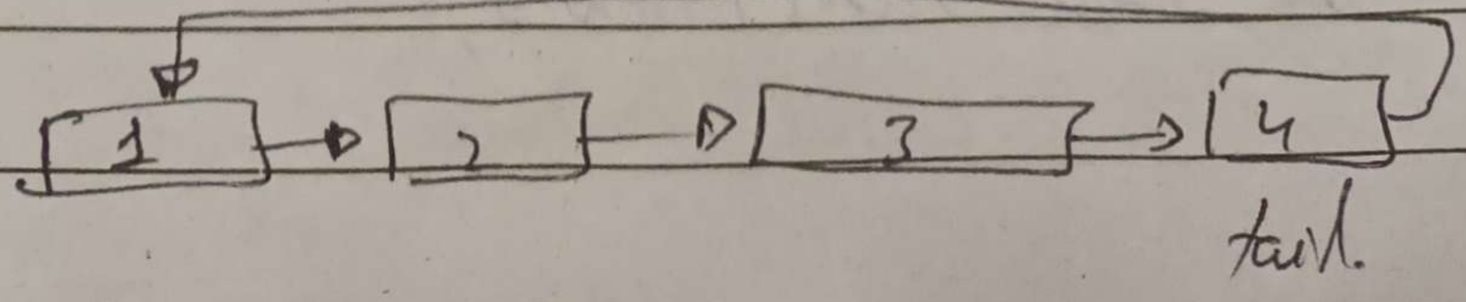
```
class Node {
    int data;
    Node* next;
    Node(int val) { data = val; next = nullptr; }
};

class Queue {
    Node* tail;
    Queue() { tail = nullptr; }

    void enqueue(int item) {
        Node* newNode = new Node(item);
        if (!tail) { // Que empty
            tail = newNode;
            tail->next = tail;
        } else {
```

DATE: __/__/____

```
        NewNode->next = tail->next;
        tail->next = newNode;
        tail = newNode;
    }
}

int dequeue() {
    if (!tail) {
        cout << "Queue Empty";
        return -1;
    }
    if (tail->next == tail) { // one-element
        int item = tail->data;
        delete tail;
        tail = nullptr;
        return item;
    } else {
        Node* front = tail->next;
        int item = front->data;
        tail->next = front->next;
        delete front;
        return item;
    }
}
```

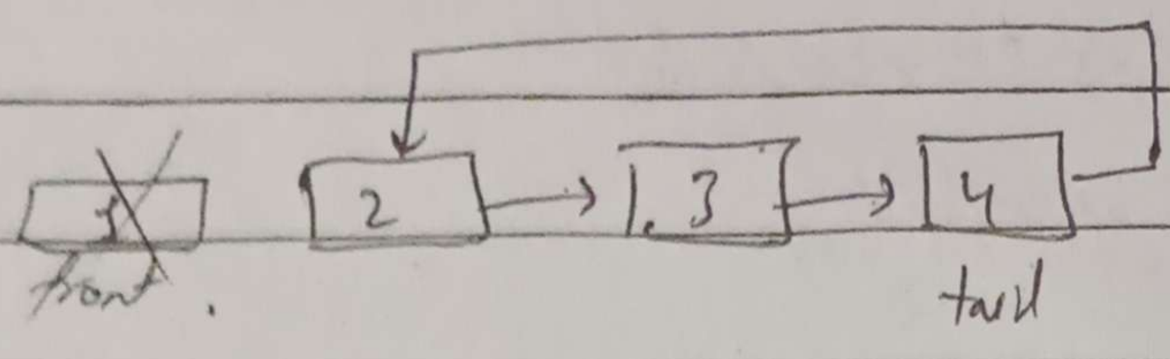


Insertion:-



DATE: __/__/____

Dequeue:-



Question 15
Deque

```
#define SIZE 20
class Deque{
    int item[SIZE], front, rear;
    Deque() { front = -1; rear = -1; }

void push(int x){          // insert at start
  if(front==-1){
    front=0;
    item[++rear]=x;
  }else if(front!=0){
    items[--front]=x;
  }else{
    cout << "Insertion not possible";
  }
}

int pop(){                 // remove from front
  if(front==-1){
    return -1;
  }else{
    int items=items[front];
```

DATE: __/__/____

```
        if (front == rear) {
            front = rear = -1;
        } else {
            front = front + 1;
        }
        return item;
    }
}

void Inject(int x) {        // insert at rear
    if (rear >= SIZE - 1) {
        cout << "Insertion not possible";
    } else {
        if (front == -1) {
            front++;
            rear++;
        } else {
            rear = rear + 1;
        }
        items[rear] = x;
    }
}

int Eject() {
    if (front == -1) { return -1; }
    else { int item = items[rear];
        if (front == rear) {
            front = rear = -1;
        } else { rear = rear - 1; }
        return item;
    }
```