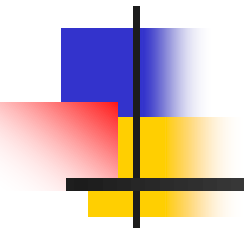
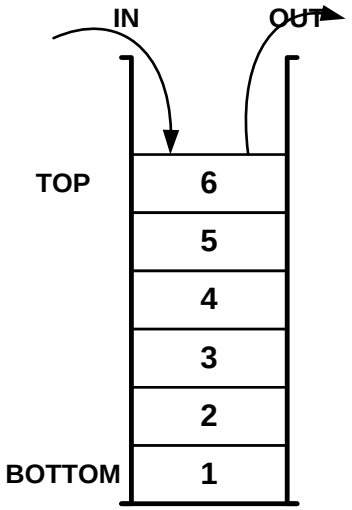


Chapter 04

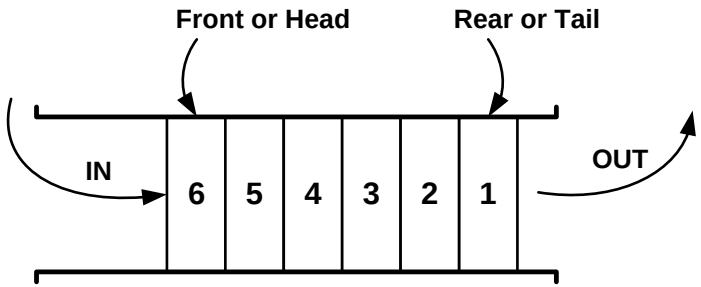
Stacks and Queues

STACKS & QUEUES

- โครงสร้างสำหรับเก็บข้อมูลชั่วคราว(ช่วยจำ) เพื่อรอการทำงาน(ให้บริการ)
- แตกต่างกันที่ลำดับการนำข้อมูลที่ฝากไว้ ออกมาใช้(ต้องใช้ให้หมด)
  - **Stacks Last In First Out (LIFO)** ข้อมูลที่เข้าทีหลังจะถูกเรียกออกมาก่อน
  - **Queues First In First Out (FIFO)** ข้อมูลที่เข้าก่อนจะถูกเรียกออกมาก่อน
    - **Queue** ที่มีการจัดลำดับความสำคัญของข้อมูลเข้า เรียกว่า **Priority Queue** ข้อมูลที่มีความสำคัญมากกว่า สามารถแทรกในตำแหน่งที่เหมาะสมได้



STACKS
Last In First Out (LIFO)



QUEUES
First In First Out (FIFO)

Operations	STACKS	QUEUES
Input	PUSH	Add/Insert/Enqueue
Output	POP	Delete/Remove/Dequeqe

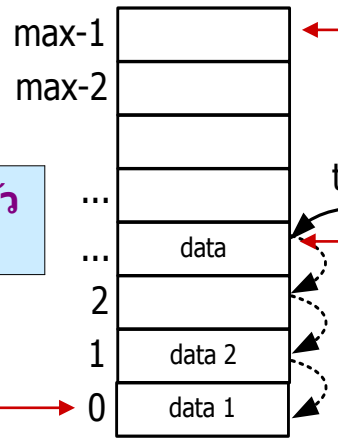
Stack Array

- การสร้าง Stack ด้วยอาร์เรย์
- Global Declarations

```
#define max 50
double stack[max];
int  top= -1;
```

```
public class Teststack {
max = 50;
double  [] stack = new double[50];
int top = -1;
.....
public static void main(String[] args) {
   Teststack  st = new Teststack();
   st.push(25.5);
   ..............
}
}
```

ใช้ตัวแปร top เป็นตัวจัดการตำแหน่งข้อมูล เริ่มต้นไม่มีข้อมูลให้ top = -1



ต้องระวังไม่ให้เก็บข้อมูลเกินตำแหน่งนี้ (over flow)

ถ้าจะเพิ่มข้อมูล ให้เพิ่มค่า top+1 ก่อน แล้วจึงนำข้อมูลไปเก็บ

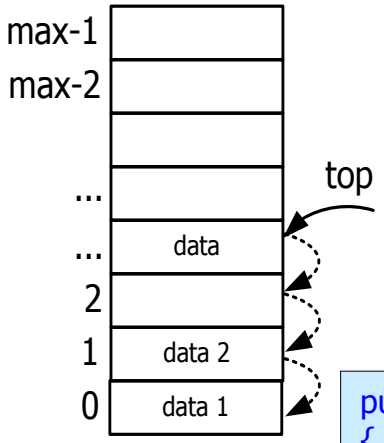
ข้อมูลตัวล่าสุดจะเก็บอยู่ในตำแหน่งที่ top

ถ้าดึงข้อมูลออกออกแล้ว ให้ลดค่า top ลง 1

ข้อมูลตัวแรกเก็บที่ top = 0

Stack Operations

- ตัวอย่าง operation ที่เกี่ยวกับ stack




```
int isEmpty()  // check stack  empty
{  if (top<0) return 0
      else return 1;
}
int isFull()              // check stack full
{  if (top >= max) return 1;
       else return 0;
}
int size()                // check stack full
{  return top+1;
}
void push(double item)          // push item to stack
{    stack[++top] = item; // if (top<max) stack[++top] = item;
}
double pop()                 //  pop item from stack
{    return  stack[top--] ; // if (top>0) return item[top--];
}
double peek()           // peek value at stack top
{   return stack[top];
}
```

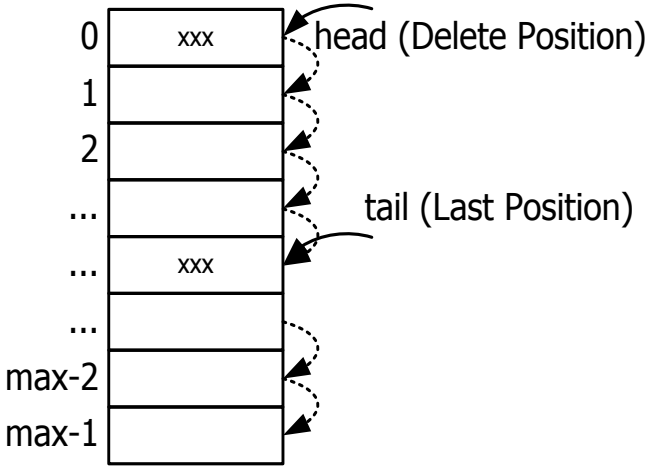
```
public boolean isEmpty()
{  if (top<0) return false;
      else return true;
}
public static boolean isFull()
{  if (top >= max) return true;
       else return false;
}
public static int size()
{  return top+1;
}
public static void push(double item)
{    stack[++top] = item;
}
public static double pop()
{    return  stack[top--] ;
}
public static double peek()
{   return stack[top];
}
```

Queue Array

- การสร้าง Queue ด้วยอาร์เรย์
- Global Declarations

```
#define max 50
typedef  double q_array[max]  ;
q_array  queue;
int  head=0, tail=-1;
```

ใช้ tail เป็นตัวชี้ตำแหน่งข้อมูลตัวสุดท้าย ต้องเพิ่มตำแหน่ง tail +1 ก่อนเก็บข้อมูล เริ่มต้นไม่มีข้อมูลต้องให้ tail = -1




```
public class Testqueue {
   max = 50;
   double  [] queue = new double[50];
   int head = 0, tail=-1;
.....................
public static void main(String[] args)
{
    Testqueue  q = new Testqueue();
    q.add(25.5);
    ..................
}
```

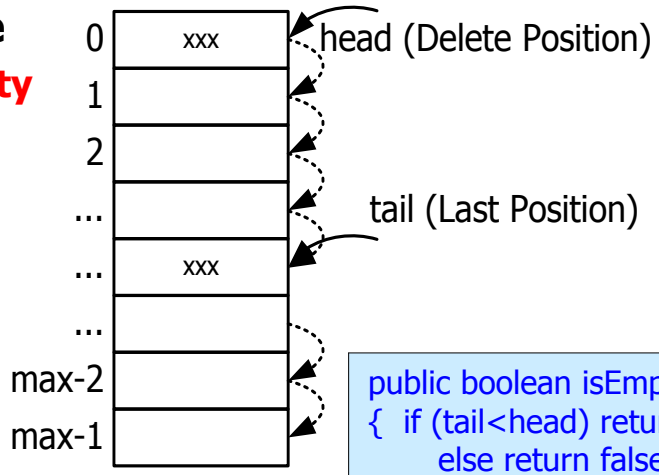
ใช้ head เป็นตัวชี้ตำแหน่งที่จะดึงข้อมูลออก เมื่อดึงข้อมูลออกแล้ว ต้องเพิ่มค่า head +1 ข้อมูลตัวแรกที่จะดึงออก head = 0 และค่า tail จะต้อง >= 0

ไม่ถูกใช้งานอีก จะทำให้อาร์เรย์เต็ม

Queue Operations

- ตัวอย่าง operation ที่เกี่ยวกับ queue

```
int isEmpty()          // check queue  empty
{  if (tail<head) return 1;
     else return 0;
}
int isFull()           // check queue full
{  if (tail < max-1) return 0;
     else return 1;
}
int size()             // check no. of data
{  return tail-head+1;
}
void add(double item)// add item to queud
{    queue[++tail] = item; // if  (tail<max-1)  queue[++head] =
}
double delete()        //  delete item from queue
{    return  queue[head++] ;  // if  (head<=tail) return queue[h
}
double peek()          // peek value at first queue
{  return queue[head];
}
```

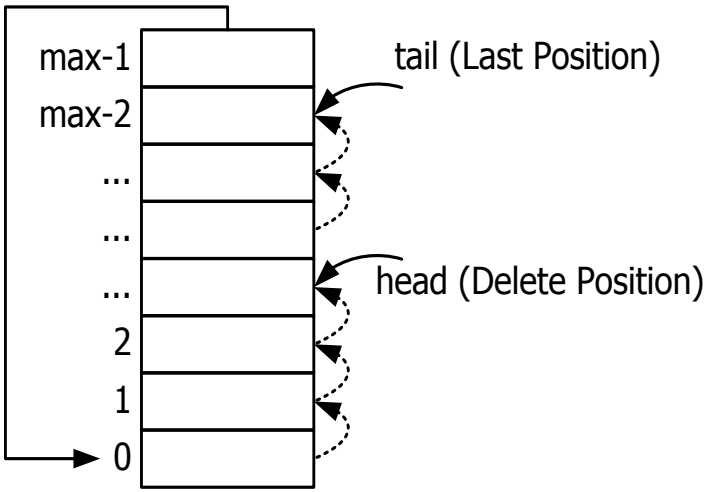


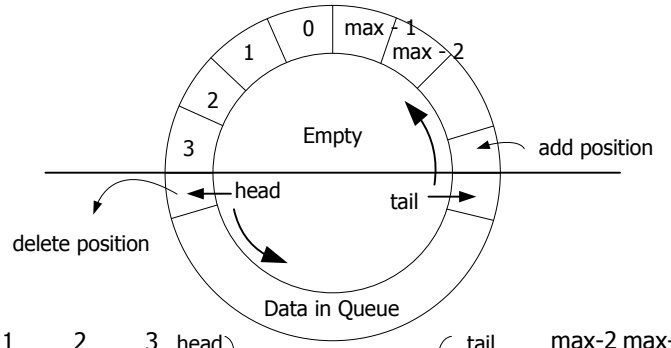

```
public boolean isEmpty()
{  if (tail<head) return true;
     else return false;
}
public boolean isFull()
{  if (tail< max-1) return false;
     else return true;
}
public int size()
{  return tail-head+1;
}
public  void add(double item)
{    queue[++tail] = item;
}
public  double delete()
{    return  queue[head++] ;
}
public  double peek()
{  return queue[head];
}
```

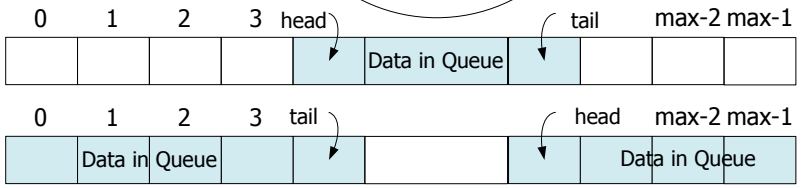
- ใช้ไปเรื่อยๆ มีโอกาสที่อาร์เรย์จะเกินขอบเขต ต้องหาทางนำเนื้อที่กลับมาใช้ใหม่

Circular Queue

- สมมุติให้ Queue มีลักษณะเป็นวงกลม







เพิ่มตำแหน่ง **tail +1** ก่อนเก็บข้อมูล
เพิ่มตำแหน่ง **head +1** เมื่อดึงข้อมูลออกแล้ว

สมมุติ **Queue** มีลักษณะเป็นวงกลม
เมื่อ **tail** หรือ **tead** เลื่อนมาเกินค่าสุดท้าย ให้เลื่อนไปที่ตำแหน่ง **0** ใหม่ ทำให้นำตำแหน่งที่ว่างกลับมาใช้ซ้ำได้
ถ้า **head = tail** แสดงว่าเหลือข้อมูลตัวเดียว
ถ้า **head < tail** แสดงว่าข้อมูลอยู่ระหว่าง
[head] .. [tail]
ถ้า **head > tail** แสดงว่าข้อมูลอยู่ระหว่าง
0 .. [tail] หรือ [head] .. [max-1]

มีโอกาสที่จะเกิด
underflow คือดึงข้อมูลหมดจนเกินที่มีอยู่
overflow คือเพิ่มข้อมูลจนล้น
แก้ปัญหาด้วยการเพิ่มตัวนับข้อมูล เพื่อตรวจสอบจำนวนข้อมูลที่มีอยู่

Linked List Imple

- การสร้าง Stack หรือ Queue ด้วย Linked List
- Global Declarations

```
typedef  struct node_tag  { double   info;
                            struct node_tag  *next; } node_type;

node_type  *first, *last, *ptr;
```

- Initial Condition

  first = last = NULL ;

- STACK Operation
  - push (insert at first node)
  - pop (delete first node)
  - .....
- Queue Operation
  - add (insert at last node)
  - remove (remove first node)
  - .....

```
public class Linkednode {
  double info;
  Linkednode next;
  public Linkednode(double num)
  {   info = num;
      next = null;
      }
}
```

```
public class LinkedListnode {
  Linkednode  first, last;
  public LinkedListnode() {
       first=null;
       last=null;
  }
 public void list_stack()
  {  Linkednode ptr;  //
     int count = 0;
     ptr  =  first;  //เริ่มที่ตัวแรก
     while (ptr != null)
     {  count++;  // นับจำนวน
        System.out.printf ("stack[%d] = %f\n", count, ptr.info);
        ptr  =  ptr.next; }   //เลื่อน ptr เป็นตัวถัดไป
  }
  ......................
  public static void main(String args[]) {
     LinkedListnode  teststack = new LinkedListnode();
     teststack.push(20);
     teststack.list_stack();
     ..............
}
```

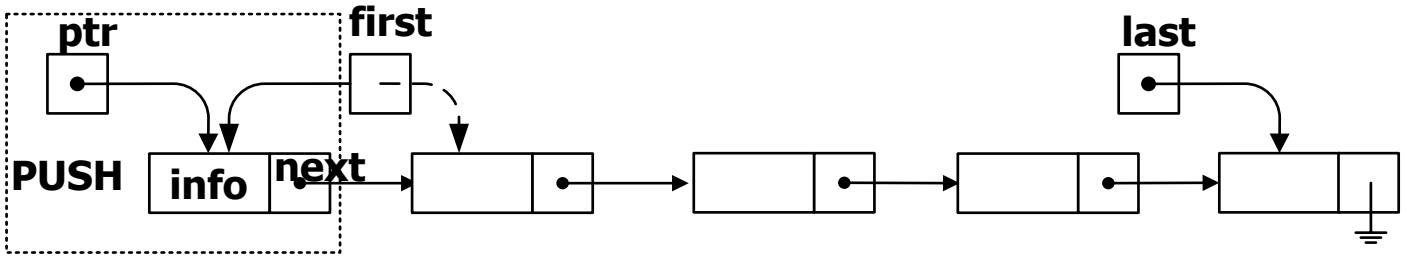
PUSH item into stack

- PUSH item into stack

```
void  push (double item)
{ if ((ptr = (node_type *) malloc (sizeof(node_type))) == NULL)
      {  printf("memory over flow\n");
       }
  else { ptr -> info = item;
         ptr -> next = first;
         first = ptr;
       }
}
```

ใช้ Linked List ทำ STACK ให้ PUSH ข้อมูล ในตำแหน่ง first

```
public  void  push(double item)  {
   Linkednode  ptr = new Linkednode(item);
   ptr.next = first;
   first = ptr;
}
```



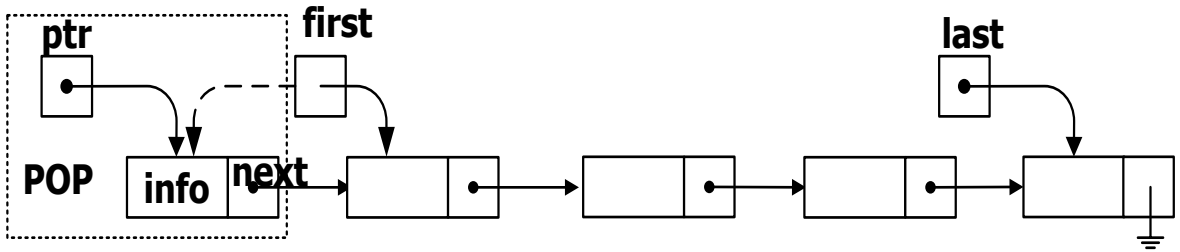
POP item from stack

- POP item from stack

```
double pop ()
 {    double item;
      ptr = first;
      item = ptr-> info;
      first = first -> next
      return  item;   }
 }
```

ใช้ Linked List ทำ STACK ให้ POP ข้อมูลจากตำแหน่ง first

```
public  double  pop() {
      Linkednode  ptr = first;
      first = first.next;
      return ptr.info ;
}
```

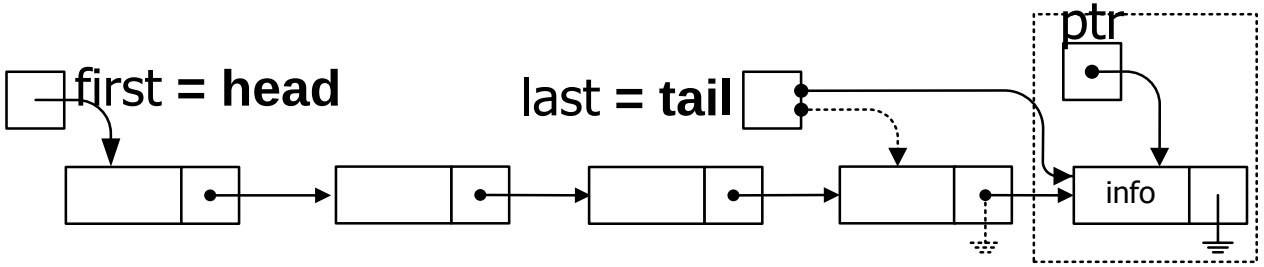


Add Queue in Linked list

```
int add (double item)  /* Add Item to last node of linked list */
{ if ((ptr = (node_type *) malloc (sizeof(node_type))) == NULL)
    return 0;
  else
 {  ptr -> info = item;
    ptr -> next = NULL;
    if  (head == NULL)
      {    head = ptr;
           tail = ptr; }
    else  { tail -> next = ptr;
            tail = ptr;  }
    return 1;
 }
}
```

ใช้ Linked List ทำ Queue ให้ Add ข้อมูลต่อท้ายตำแหน่ง last (tail)

```
public  void  add (double item)
{  Linkednode  ptr = new  Linkednode(item);
   if  (head == null)
      {  head = tail =  ptr; }
   else  { tail.next = ptr;
           tail = ptr;  }
}
```

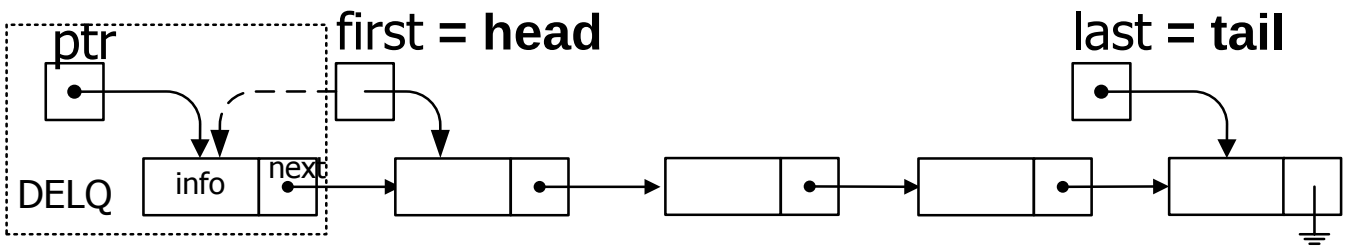


DELETE Queue in Linked list

```
double remove () /* Delete first  node of linked list */
 { double item;
    ptr = head;
          item = ptr-> info;
          head = head -> next ;
          free(ptr);  }
 }
```

ใช้ Linked List ทำ Queue ให้ Delete ข้อมูลจากตำแหน่ง first (head)

```
public double remove() {
    Linkednode  ptr = head;
    head = head.next;
     return  ptr.info;
}
```



Class : Stack & Queue Java

- ตัวอย่าง การใช้คลาส Stack ของ Java

```
import java.util.stack;
        static Stack<String> STACK = new Stack<String>();
        STACK.clear();
        STACK.push(buff);
        buff = STACK.peek();
        buff = STACK.pop();
        int size = STACK.size();
```

- ตัวอย่าง การใช้คลาส Queue ของ Java

```
import java.util.Queue;
        static Queue<String> QUEUE = new Queue<String>();
        QUEUE.clear();
        QUEUE.add(buff);
        buff = QUEUE.peek();
        buff = QUEUE.remove ();
        int size = QUEUE.size();
```

- ตัวอย่าง การนำคลาส LinkedList ใช้เป็น stack/Queue

```
import java.util.LinkedList;
        static LinkedList<String> STQ = new LinkedList<String>();
        STQ.clear();
        STQ.addFirst(buff); // ใส่ข้อมูลที่ตัวแรกสำหรับ push stack
        STQ.add(buff); // ใส่ข้อมูลที่ตัวสุดท้ายสำหรับ add queue
        buff = STQ.peekFirst(); // peek ดูข้อมูลที่จะดึง
        buff = STQ.removeFirst(); // pop/remove ดึงข้อมูลตัวแรก
        int size = STQ.size();
```

Stack/Queue Applications

- การเขียน Arithmetic Expressions
  - การเขียนนิพจน์คณิตศาสตร์โดยทั่วไปจะเขียนโดยให้ตัวดำเนินการ(Operator) อยู่กลางระหว่างตัวถูกดำเนินการ(Operand) เรียกว่า Infix Notation
  - Infix form /* operator อยู่ตรงกลาง ต้องคำนึงถึงลำดับความสำคัญ */

2*3	2*3+5	2+3*5	2*(3+5)	2+3*(5-7)

  - การเขียนนิพจน์โดยให้ Operator อยู่หน้า Operand เรียกว่า Prefix Notation หรือ Polish Notation
  - Prefix form /* operator อยู่ข้างหน้า*/

*,2,3	+,*,2,3,5	+,2,*,3,5	*,2,+,3,5	+,2,*,3,-,5,7

  - การเขียนนิพจน์โดยให้ Operator อยู่หลัง Operand เรียกว่า Postfix Notation หรือ Reverse Polish Notation
  - Postfix form /* operator อยู่ข้างหลัง*/

2,3,*	2,3,*, 5, +	2,3,5,*,+	2,3,5,+,*	2,3,5,7,-,*,+

- การเขียนสมการในรูป prefix หรือ postfix form ทำให้ไม่ต้องกังวลเรื่องลำดับความสำคัญและเครื่องหมาย ( ) เนื่องจากมีการปรับเปลี่ยนตำแหน่งของ operator ตามลำดับความสำคัญก่อนหลังของการคำนวณ ทำให้สามารถคำนวณนิพจน์คณิตศาสตร์ตามลำดับที่ถูกต้องได้
- การหาคำตอบของนิพจน์ prefix หรือ postfix ต้องอาศัย stack และ queue มาช่วย

transform infix to postfix

- การแปลงสมการ infix ให้อยู่ในรูป postfix

1. ใส่ "(" และ ")" คร่อม infix expression //ไม่ใส่ก็ได้ แต่ต้องเช็คstack(4) ตอนหมดข้อมูล
2. ดึงข้อมูล (TOKEN) ออกจาก string ทีละชุด จากซ้ายไปขวา
   - 2.1 ถ้าเป็น "(" ให้ PUSH ใส่ใน STACK (String)
   - 2.2 ถ้าเป็น ตัวเลข number ให้ส่งไปยัง output
   - 2.3 ถ้าเป็น operator ให้พิจารณาดังนี้
     - 2.3.1 POP operators ทุกตัวที่มีลำดับความสำคัญมากกว่าหรือเท่ากับ operator ที่พบใน 2.3 ส่งไปยัง output
     - 2.3.2 PUSH operator ที่เจอใน 2.3 ใส่เข้าไปใน STACK แทน
   - 2.4 ถ้าเป็น ")" ให้ POP operators ทุกตัวจาก STACK ส่งไปยัง output จนกว่าจะเจอ "("
     // pop "(" ออกมา แต่ไม่ต้องส่งไป output
3. ทำซ้ำในข้อ 2 จนกว่าข้อมูลจะหมด // ข้อมูลจะหมดที่ ")" ที่เพิ่มเข้าไป
4. ถ้ามีข้อมูลเหลืออยู่ใน stack ให้ POP ข้อมูลทั้งหมดออกมาด้วย //กรณีที่ไม่ได้เพิ่ม ")"

```
else if (group == 7) // ( open parenthesis
    oprstack.push(token[i]);
```

```
else if (group == 8) // ) close paren
    { do { buff = oprstack.pop();
           if (!buff.equals("("))
               postfix.add(buff);
       } while (!buff.equals("("));
    }
```

```
if (group == 1) // Number
    postfix.add(token[i]);
```

```
static ArrayList<String> postfix = new ArrayList<String>();
static Stack <String> oprstack = new Stack<String>();
public void Change_infix_to_postfix(String[] token) {
    int group, cur, prior = 0, i;
    String buff;
        postfix.clear();
        oprstack.clear(); for (i = 0; i < token.length; i++) {
            group = TokenAnalysis.check_group(token[i]);
             ..........................
        }
}
```

```
else if (group >= 2 && group <= 6) // operator&function
{  do { cur = TokenAnalysis.check_group(token[i]);
        buff = oprstack.peek();
        prior =TokenAnalysis.check_group(buff);
        if (prior >= cur && prior <= 6) {
            buff = oprstack.pop();
            postfix.add(buff);
        }
   } while (prior >= cur && prior <= 6);
            oprstack.push(token[i]);
}
```

Change 2+4*5/(3+7)-8 to postfix

step	Expression	Stack	Output string (queue)
1	(2+4*5/(3+7)-8)	NULL	NULL
2.1	(2+4*5/(3+7)-8)	(	NULL
2.2	2+4*5/(3+7)-8)	(	2← (สมมุติลูกศรคือตำแหน่ง head ของ queue)
2.3.2	+4*5/(3+7)-8)	(+	2←
2.2	4*5/(3+7)-8)	(+	2, 4← (สมมุติให้,คั่นระหว่างข้อมูลแต่ละตัว)
2.3.2	*5/(3+7)-8)	(+,*	2, 4←
2.2	5/(3+7)-8)	(+,*	2, 4, 5←
2.3	/(3+7)-8)	(+,/	2, 4, 5, *←
2.1	(3+7)-8)	(+,/,(	2, 4, 5, *←
2.2	3+7)-8)	(+,/,(	2, 4, 5, *, 3←
2.3.2	+7)-8)	(+,/,(+	2, 4 ,5, *, 3←
2.2	7)-8)	(+,/,(+	2, 4, 5, *, 3, 7←
2.4	)-8)	(+,/	2, 4, 5, *, 3, 7, +←
2.3	-8)	(-	2, 4, 5, *, 3, 7, +, /, +←
2.2	8)	(-	2, 4, 5, *, 3, 7, +, /, +, 8←
2.4	)	NULL	2, 4, 5, *, 3, 7, +, /, +, 8, -←

Postfix Calc

- การหาคำตอบของสมการ postfix

1. ดึงข้อมูล (TOKEN) ออกจากสมการทีละตัวจากซ้ายไปขวา
   - 1.1 ถ้าเป็น ตัวเลข ให้ PUSH ใส่ใน STACK (STACK ของเลขจำนวนจริง)
   - 1.2 ถ้าเป็น unary operator ให้ POP ตัวเลข 1 ตัว จาก STACK มาคำนวณตามคำสั่ง แล้วนำคำตอบที่ได้ PUSH ใส่กลับเข้าไปใน STACK
   - 1.3 ถ้าเป็น binary operator then POP ตัวเลขออกจาก STACK 2 ตัวมาคำนวณตามคำสั่ง (POPตัวหลัง operate POPตัวแรก) แล้วนำคำตอบที่ได้ PUSH ใส่กลับเข้าไปใน STACK
2. ทำซ้ำ ในข้อ 1 จนกว่าข้อมูลจะหมด
3. POP ตัวเลขจาก STACK (เหลือตัวเดียว) มาเป็นคำตอบ

```
if (group == 1) { // number
    num1 = ValueOf(token);
    numstack.push(num1);
}
```

```
public double ValueOf(String str) {
    if (str.equalsIgnoreCase("pi"))
        return Math.PI;
    else if (str.equalsIgnoreCase("E"))
            return Math.E;
    else if (str.equalsIgnoreCase("ans"))
            return ans;
    else
        return Double.parseDouble(str);
}
```

```
else if (group >= 2&&group<=4) { //bi. operator
        num1 = numstack.pop();
        num2 = numstack.pop();
        if ((group==2) && token.equals("+"))
            numstack.push(num2 + num1);
        else if  .....
}
```

```
static Stack <Double> numstack = new Stack<Double>();
public double Calculate_postfix() {
double ans = 0, num, num1, num2;
int i, group;
String token = new String("");
    for (i = 0; i < postfix.size(); i++) {
        token =postfix.get(i);
        group = TokenAnalysis.check_group(token);
        if .....................
    }
    ans = numstack.pop(); // output answer
    return ans;
}
```

```
else if (group == 5) { // negative sign
        num1 = numstack.pop();
        numstack.push(-num1);
}
```

```
else if (group == 6) { // function
        if (token.equalsIgnoreCase("sin"))
          { num = Math.sin(numstack.pop() * Math.PI / 180);
            numstack.push(num);
          }
        else if .................
}
```

calculate 2,4,5,*,3,7,+,/,+,8,-

Equation	Operation	Stack
2,4,5,*,3,7,+,/,+,8,-	PUSH(2), PUSH(4), PUSH(5)	2, 4, 5 ←
*,3,7,+,/,+,8,-	POP(5), POP(4),*, PUSH(20)	2, 20 ←
3,7,+,/,+,8,-	PUSH(3), PUSH(7)	2, 20, 3, 7 ←
+,/,+,8,-	POP(7), POP(3),+, PUSH(10)	2, 20, 10 ←
/,+,8,-	POP(10), POP(20), / , PUSH(2)	2, 2 ←
+,8,-	POP(2), POP(2),+ , PUSH(4)	4 ←
8,-	PUSH(8)	4, 8 ←
-	POP(8), POP(4), - , PUSH(-4)	-4 ←
NULL	POP(-4) to answer	NULL

Evaluating priority of operators

- ตัวอย่างการกำหนดลำดับความสำคัญของ operator

Operators/Token	type/Priority
(,) (parenthesis)	8
function (function call-program)	7
all unary arithmetic operators (sign)	6
x , / , div , mod (Multiplicative)	5
+ , - (Additive)	4
== , != , < , > , <= , >= (Relational)	3
! (Logical NOT)	2
&& (Logical AND) , || (Logical OR)	1
= (Assign Statement)	0