.

给出循环队列实现，入队，出队

```
public void enqueue(Object x) throws Overflow{  /*入队*/
        if(isFull()){
                throws new Overflow();
        }
        else{
                length++;
                rear = (rear+1)%m;
                Q[rear] = x;
        }
}

public Object dequeue() throws Underflow{  /*出队*/
        if(isEmpty()){
                throws new Underflow();
        }
        else{
                length--;
                return Q[(rear-length+m)%m];
        }
}
```

二叉树的机内存储（表示）：广义表表示 ， 双亲表示法 ， 左子女—右兄弟表示法

森林的后根遍历与对应的二叉树中序遍历一致

非递归中序遍历

Inorder non-recursive algorithm

```
void Inorder(BinaryNode <T> * t)
{ Stack<BinaryNode<T>*> s(10);
  BinaryNode<T> * p = t;
  for ( ; ; )
  { 1) while(p!=NULL)
       { s.push(p);   p = p->Left; }
     2) if (!s.IsEmpty( ))
       { p = s.pop( );
         cout << p->element;
         p = p->Right;
       }
       else return;
  }
}
```

排序算法的稳定性：
如果待排序的对象序列中，含有多个关键码值相等的对象，用某种方法排序后，这些对象的相对次序不变的，则是稳定的,否则为不稳定的。

复习例题---在O(n)时间内实现将负数排在所有非负数之前。

```
void sort ( float [ ] a, int n )
{  int i = 0 ,  j = n-1 ;
   while ( i != j )
   {  while ( a[j] >= 0.0 && i < j ) j-- ;
      while ( a[i] < 0 && i < j ) i++ ;
      float temp = a[i] ;  a[i] = a[j]; a[j] = temp;
      j-- ; i++ ;
   }
}
```

信号量解决读者写者问题-写者优先

```
semaphore rmutex, wmutex, S;
   rmutex=1; wmutex=1; S=1; //增加互斥信号量S
int readcount=0; //读进程计数
```

```
process reader_i( ) {
while (true) {
   P(S);
  P(rmutex);
   if (readcount==0) P(wmutex);
      readcount++;
  V(rmutex);
  V(S);
   读文件;
  P(rmutex);
      readcount--;
   if(readcount==0)  V(wmutex);
  V(rmutex);
  }
}
```

```
process writer_i( ) {
  while(true) {
   P(S);
    P(wmutex);
    写文件;
    V(wmutex);
   V(S);
  }
}
```

读者/写者问题(写者优先)

```
int readcount = 0, writecount = 0;
semaphore x=1, y=1, z=1;        // readcount,writecount互斥
semaphore rmutex=1,wmutex=1;        // 读锁，写锁
```

```
process reader
{
 P(z);
  P(rmutex);
   P(x);
    readcount++;
    if (readcount==1) P(wmutex);
   V(x);
 V(rmutex);
 V(z);
   read;
 P(x);
   readcount--;
   if (readcount==0) V(wmutex);
 V(x)
};
```

```
process writer
{
 P(y);
   writecount++;
   if (writecount==1) P(rmutex);
 V(y);
  P(wmutex);
    write;
  V(wmutex)
 P(y);
   writecount--;
   if (writecount==0) V(rmutex);
 V(y);
};
```

所有边权均不相同的无向图最小生成树是唯一的

a) 检测回文单词

- 比较头尾两个字符，相同则向中间聚拢比较，不相同则返回false。
  - 边界条件和递归返回段：
    - 要比较的子字符窜为空，返回true
  - 递归前进段：截取当前字符串从位置1到位置length-1的子字符串，返回其回文性。

```
public static boolean palindrome0(String word, int low, int high) {
  if (low > high)
    return true;
  if (word.charAt(low) == word.charAt(high)
  || Math.abs(word.charAt(low) - word.charAt(high)) == 32)
    return palindrome(word, low+1, high-1);
  else
    return false;
}
```

b) 检测回文句子

- 在递归过程中去除不符合要求的字符

```
public static boolean palindrome(String word, int low, int high) {
  if (low > high)
    return true;
  while (!Character.isLetter(word.charAt(low)))
    low++;
  while (!Character.isLetter(word.charAt(high)))
    high--;
  if (word.charAt(low) == word.charAt(high)
      || Math.abs(word.charAt(low) - word.charAt(high)) == 32)
    return palindrome(word, low+1, high-1);
  else
    return false;
}
```

编写一个非递归方法以 O（n）的时间复杂度反转单链表

代码

```
public void reverse() {
        ListNode p1 = header.next;
        if (p1 == null)       return;
        ListNode p2 = p1.next;
        if (p2 == null)       return;
        ListNode p3 = p2.next;
        p1.next = null;   //将firstNode.next设为null

        // node.next设为前一个node ,然后指针前移一个位置
        while (p3 != null) {
                p2.next = p1;
                p1 = p2;
                p2 = p3;
                p3 = p3.next;
        }
        p2.next = p1; //最后一个元素指向倒数第二个元素
        header.next = p2;  //将头指针指向最后一个元素
    }
```

循环左移算法

```
void Converse(int R[],int n,int p){
    Reverse(R,0,p-1);
    Reverse(R,p,n-1);
    Reverse(R,0,n-1);
}

void Reverse(int R[],int from,int to) {
    int i,temp;
    for(i = 0; i < (to-from+1)/2; i++)
    {
        temp = R[from+i]; R[from+i] = R[to-i]; R[to-i] = temp; }
    }
}
```

- 1）统计二叉树中叶结点的个数。

```
public static int leafNum(BinaryNode root) {
    if (root == null)
        return 0;
    if (root.left == null && root.right == null)
        return 1;
    return leafNum(root.left) + leafNum(root.right);
}
```

- 2）以二叉树为参数，交换每个结点的左子女和右子女

```
public static void switchLR(BinaryNode root) {
    if (root == null)
        return;
    BinaryNode tmp = root.left;
    root.left = root.right;
    root.right = tmp;
    switchLR(root.left);
    switchLR(root.right);
}
```

写一递归函数实现在带索引的二叉搜索树(IndexBST)中查找第 k 个小的元素

```
Public BinaryNode findkth(BinaryNode t,int k)
{
    if(k<=0 || t==null)
        return null;
    if(k<t.leftsize)
        return findkth(t.left ,k);
    else if(k>t.leftsize)
        return findkth(t.right ,k-
        t.leftsize);
    else
        return t;
}
```

对于一个有N 个结点的AVL 树，其最大高度是多少？最小高度是多少？

∵费波那契数树是具有相同高度的所有平衡二叉树中结点个数最少的，

$$n+1 \geqslant N_h+1=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}+0(1)$$

$$\therefore h \leqslant \frac{1}{\log_2\frac{1+\sqrt{5}}{2}}\log_2(n+1)+0(1)\approx\frac{3}{2}\log_2(n+1)$$

这就是h的最大值；

至于h的最小值，是一颗完全二叉树的时候，高度最小，有

$2^{h+1}-1=n$，所以$h=\log_2(n+1)-1$.

如果 str 是“abc”，那么输出的串则
是 abc，acb，bac，bca，cab，和 cba。

代码

```
void permute(char [] str, int low, int
    high){//low=0,high=str.length-1
     if(low==high) {
        System.out.println(str);
     } else {
         for(int i=low; i<=high; i++){
              char temp;
              temp = str[i]; str[i]  = str[low]; str[low] = temp;
       permute(str, low+1, high);
              temp = str[i]; str[i]  = str[low]; str[low] = temp;
          }//end of for
    }
}
```

递归求解链表长度

代码

```
public class Length{
    public static int length = 0;
    public static int GetLength(Iterator it){
          if(!it.hasNext())
                    return 0;
          else{
                    it.next();//遍历
                    return 1+GetLength(it);
                    }
          }
```

设n为正整数，分析下列各程序段中加下划线的语句的执行次数。

2）

```
x = 0; y = 0;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j++)
        for (int k = 1; k <= j; k++)
            x = x+y;
```

$$f(n)=\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\frac{1}{6}n(n+1)(n+2)$$

```
public  static int findMax(int[]  a, int n){
//n表示第n个元素，它在数组中位于a [n-1]处
   if(n==1){
            return a [0];
      }
      else{
         int temp=findMax(a,n-1);
            return temp>a [n-1]?temp:a  [n-1];
      }
}
```

Special Matrix

3) Tridiagonal

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & a_{33} & a_{34} & \\ & \cdots\cdots & \cdots\cdots & \cdots\cdots & \\ & & & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

Location mapping in row-major order:

$$Loc(a(i,j)) = Loc(a(1,1)) + [(i-1)*3 - 1 + (j-i+1)]*l$$

6.3 Heaps

```
private void percolateDown( int hole )
{  int child;
   Comparable tmp = array[ hole ];
   for( ; hole *2 <= currentSize; hole = child )
   {  child = hole * 2;
      if ( child != currentSize &&  array[ child + 1 ].compareTo( array[ child ] ) < 0 )
            child++; if( array[child ].compareTo( tmp ) < 0 )
         array[ hole ] = array[ child ];
               else
         break;
    }
    array[ hole ] = tmp;
}
```

16 14

最小根 percUp

```
private static void percUp( Comparable [ ] a, int start )
{   int j = start,   i = j / 2;
    Comparable temp = a [j];
    while (j > 1)
      { if ( a[i] <= temp) break;
        else { a[j] = a[i];   j= i;   i = i / 2; }
      }
    a[j] = temp;
}
```

非递归先序访问

借助一个栈，因为每次都是栈顶出栈，即栈顶都是先访问的节点，先序遍历的思想是先根，再左孩子，再右孩子。

故访问完当前节点后，应该先将右孩子入栈，再左孩子入栈即可。

```
void PreOrder1(BTNode *b)
{
    BTNode *St[MaxSize], *p;
    int top = -1;
```

```
    if (b!=NULL)
    {
        top++;
        St[top] = b;
        while(top > -1)      //the stack is not empty then loop
        {
            p = St[top];
            top--;
            printf("%c", p->data);
            if(p->rchild) St[top++] = p->rchild;
            if(p->lchild) St[top++] = p->lchild;
        }
    }
    printf("\n");
}
```

层次遍历

```
void TravLevel(BTNode* b)
{
    BTNode *Qu[MaxSize];
    int front, rear;
    front = rear = 0;
    if( b!= NULL)
        printf("%c", b->data);
    rear++;
    Q[rear]= b;
    while(rear != front)
    {
        front= (front+ 1)%MaxSize;        //front head come out;
        b = Qu[front];
        if(b->lchild != NULL)                    //print left child, and
enter stack
        {
            printf("%c", b->lchild->data);
            rear = (rear+1)%MaxSize;
            Qu[rear] = b->lchild;
        }
        if(b->rchild != NULL)                    //print right child, and
enter stack
```

```
        {
            printf("%c", b->rchild->data);
            rear = (rear+1)%MaxSize;
            Qu[rear]=b->rchild;
        }
    }
    printf("\n");
}
```