Information for Midterm 2

List ADT operations (pa6):

```
List();
List(const List& L);
~List();
int length() const;
ListElement front() const;
ListElement back() const;
int position() const;
ListElement peekNext() const;
ListElement peekPrev() const;
void clear();
void moveFront();
void moveBack();
ListElement moveNext();
ListElement movePrev();
void insertAfter(ListElement x);
void insertBefore(ListElement x);
void setAfter(ListElement x);
void setBefore(ListElement x);
void eraseAfter();
void eraseBefore();
int findNext(ListElement x);
int findPrev(ListElement x);
void cleanup();
List concat(const List& L) const;
std::string to_string() const;
bool equals(const List& R) const;
friend std::ostream& operator<<( std::ostream& stream, const List& L );
friend bool operator==( const List& A, const List& B );
List& operator=( const List& L );
```

Some BST Algorithms:

```
InOrderTreeWalk(x)
   if x != NIL
      InOrderTreeWalk(x.left)
      print(x.key)
      InOrderTreeWalk(x.right)

PreOrderTreeWalk(x)
   if x != NIL
      print(x.key)
      PreOrderTreeWalk(x.left)
      PreOrderTreeWalk(x.right)

PostOrderTreeWalk(x)
   if x != NIL
      PostOrderTreeWalk(x.left)
      PostOrderTreeWalk(x.right)
      print(x.key)
```

```
TreeMinimum(x) Pre: x != NIL
   while x.left != NIL
      x = x.left
   return x

TreeMaximum(x) Pre: x != NIL
   while x.right != NIL
      x = x.right
   return x

TreeInsert(T, z)
   y = NIL
   x = T.root
   while x != NIL
      y = x
      if z.key < x.key
         x = x.left
      else
         x = x.right
   z.parent = y
   if y == NIL
      T.root = z    // tree T was empty
   else if z.key < y.key
      y.left = z
   else
      y.right = z

Transplant(T, u, v)
   if u.parent == NIL
      T.root = v
   else if u == u.parent.left
      u.parent.left = v
   else
      u.parent.right = v
   if v != NIL
      v.parent = u.parent

Delete(T, z)
   if z.left == NIL               // case 1  or case 2.1 (right only)
      Transplant(T, z, z.right)
   else if z.right == NIL         // case 2.2 (left only)
      Transplant(T, z, z.left)
   else                           // case 3
      y = TreeMinimum(z.right)
      if y.parent != z
         Transplant(T, y, y.right)
         y.right = z.right
         y.right.parent = y
      Transplant(T, z, y)
      y.left = z.left
      y.left.parent = y
```